12-1993

Using Discovery-Based Learning to Prove the Behavior of an Autonomous Agent

David N. Mezera

AFIT/GCE/ENG/93D-10

USING DISCOVERY-BASED LEARNING TO IMPROVE THE BEHAVIOR OF AN AUTONOMOUS AGENT

THESIS
David N. Mezera
Captain, USAF

AFIT/GCE/ENG/93D-10

Approved for public release; distribution unlimited

AFIT/GCE/ENG/93D-10

USING DISCOVERY-BASED LEARNING TO IMPROVE THE BEHAVIOR OF AN AUTONOMOUS AGENT

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Computer Engineering)

David N. Mezera, B.S.E.E.

Captain, USAF

December 1993

Acknowledgements

This work would not have been possible without the the support of several people. I would first like to thank my advisor, Major Gregg Gunsch, for keeping me on a very long leash. He gave me enormous freedom in pursuing the research that I thought would be interesting using the approach of my choosing, but he would gently tug on the leash when I began to stray from my original objectives. I am also grateful for his timely reassurances when things seemed darkest. I would also like to thank my committee members, Dr. Eugene Santos and Dr. Henry Potoczny, for their contributions to this effort.

I owe much to my family. To my parents, George and Ruth Mezera: Thank you for nurturing the curiosity within me through the years. I've never forgotten that "the world is my oyster." I will strive to raise my own children with the same constant support and love that you both showed me. To my grandfather, George Sr.: You started me on the electrical engineering road almost sixteen years ago when you mailed me a logic probe kit and a soldering iron. I hope you can look down from Heaven and see the impact that small gift had on the course of my life. To my my wife, Stacy, and my children, Alek and Jonathon: For the last year and a half, you've had to deal with an often absent and crabby family member. Thanks for your unending patience and support during this very trying period. Stacy, you truly are the "shining joy and jewel of all my kingdom."

But most of all, I thank God for giving me this opportunity and clearing the rocky path to AFIT. Your Word gave me immeasurable support and helped me keep everything in perspective.

> When I said, "my foot is slipping," your love, O Lord, supported me. When anxiety was great within me, your consolation brought joy to my soul.
>
> *Psalm 95:18-19, NIV*

David N. Mezera

Table of Contents

List of Figures

List of Tables

Abstract

Computer-generated autonomous agents in simulation often behave predictably and unrealistically. These characteristics make them easy to spot and exploit by human participants in the simulation, when we would prefer the behavior of the agent to be indistinguishable from human behavior. An improvement in behavior might be possible by enlarging the library of responses, giving the agent a richer assortment of tactics to employ during a combat scenario. Machine learning offers an exciting alternative to constructing additional responses by hand by instead allowing the system to improve its own performance with experience.

This thesis presents NOSTRUM, a discovery-based learning (DBL) system designed to work in tandem with the MAXIM air combat simulator. Through a process of repeated experimentation modeled after the scientific method, NOSTRUM was able to discover many responses that were more appropriate than the single mode of agent control implemented in the original MAXIM program. NOSTRUM often found responses that dramatically improved the offensive position of the agent, and it sometimes placed the agent in position for an extended shot on the target when one was not available before.

USING DISCOVERY-BASED LEARNING TO IMPROVE THE BEHAVIOR OF AN AUTONOMOUS AGENT

I. Introduction

1.1 Background

Combat pilots undergo many intense hours of training to acquire the skill necessary to effectively control a modern-day fighter aircraft. While some of this training is conducted in the cockpit, a significant portion of training is conducted outside the aircraft in combat simulators. The Navy, Air Force, and Army each spend in excess of a half-billion dollars annually for training, and spending in this area is projected to increase as the military tries to maintain a quality force with a shrinking budget (14). When standardized and economical training is the goal, simulators seem to present the ideal training method. Simulators can provide identical training to multiple student pilots, something that human opponents cannot do. Each person reacts differently in similar situations, and human opponents are incapable of repeating a combat simulation exactly the same time and again.

There are serious inadequacies, however, related to the quality of training that simulations can provide. Simulators are able to generate consistent training scenarios, but often behave predictably. After several encounters against a simulated opponent, the behavior of the opponent seems decreasingly realistic to the trainee. Realism could be improved by making simulated opponents behave more intelligently, transforming them from predictable, scripted entities into *autonomous agents*. A well-programmed autonomous agent would be indistinguishable from other human players, and could be used to populate simulations with computer-generated opponents or allies. Research is underway at the Air Force Institute of Technology (AFIT) to explore artificial intelligence (AI) techniques that can accomplish this objective.

AFIT's research efforts have been in support of the Distributed Interactive Simulation (DIS) and the Computer-Generated Forces (CGF) projects, managed by the Advanced

Research Projects Agency. The goal of these projects is to connect world-wide military forces for participation in real-time combat simulations over a high-speed computer network (30). AFIT students have developed a prototype system called MAXIM, a proof of concept program for autonomous air combat agents, for potential use in the DIS. MAXIM autonomous agents could be integrated into DIS to act as adversaries or allies, filling out a simulation with additional players when simulator resources are scarce and human participants are few. Unfortunately, MAXIM agents are not exceedingly realistic.

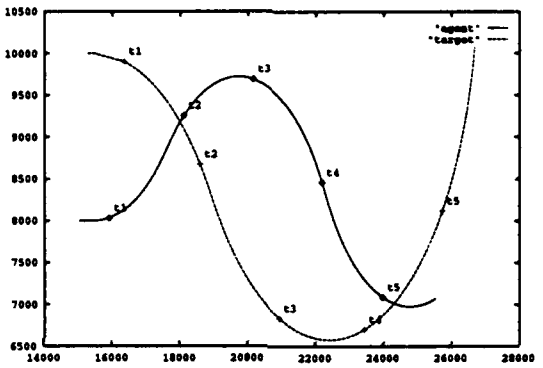

Figure 1.1 **An example of inappropriate MAXIM behavior.** In this scenario the agent overshoots the target and loses the opportunity to achieve a positional advantage.

MAXIM is capable of limited intelligent behavior. Although MAXIM does produce autonomous agents that can pursue and destroy enemy targets, only a small number of strategies are built into the system. Agents pursue each other as fast as possible within the flight envelope, sometimes turning at velocities that a human pilot could not withstand. The behavior of a MAXIM agent is also quite predictable and can be beaten by exploiting inadequacies in its responses. Figure 1.1 illustrates one instance when MAXIM agent behavior is inadequate. In this scenario, the agent and the target were initially traveling along parallel headings. The agent was traveling at lower velocity than the target aircraft and consequently had a smaller turning radius, but failed to use this difference to its advantage.

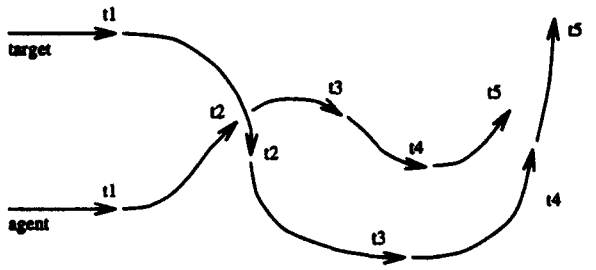

Figure 1.2 **A more appropriate MAXIM response.** Using the flat scissors, the agent could have made use of its shorter turning radius to get behind the target.

A classic maneuver known as the "flat scissors," shown in Figure 1.2, could have been used to achieve a better end-game position for the agent than is possible with the fixed type of control used by MAXIM (8, 28).

Additional strategies could be added to MAXIM by manually coding the responses into the architecture, but this approach is not very attractive for several reasons. One of the biggest detractors is that a thorough understanding of the combat tactics domain would be required before knowledge could be incorporated into MAXIM. A human expert would have to enumerate the tactics that can be used and the preconditions for each. Frequently, a knowledge engineer is needed to elicit relevant information from the expert and translate it into a form usable by a computer. Not only is this process labor intensive, but it is also prone to error because of differences in interpretation. For the combat tactics domain, acquiring all the requisite knowledge is not a simple matter. For example, the PDPC system, also developed at AFIT, used a comprehensive knowledge-base of approximately 200 rules to generate air combat agent behavior (9, 10). A possible way to circumvent the knowledge bottleneck would be to incorporate a learning capability into the system. While the agent is engaged against an opponent, or off-line during training scenarios, the agent could find out for itself, *and remember*, sets of preconditions and associated tactics based on its own experience.

Machine learning is an AI technique that can be used to improve the performance of a system by leveraging knowledge that was useful as a solution to a previously encountered

problem. Many techniques exist to implement machine learning, but each type of learning can be described as either being a *supervised* or *unsupervised* technique. Supervised learning requires the assistance of a human teacher to identify positive and negative examples of a concept, whereas unsupervised learning can proceed without any human involvement. This type of machine learning has potential for accelerating the skill acquisition process since the pace of learning in an unsupervised system is limited only by the processing speed of the computer.

Discovery-Based Learning (DBL) is one type of unsupervised machine learning. Using discovery-based learning, the system mimics the scientific method: the area to be studied is restricted to a manageable domain, data is gathered, and hypotheses are formulated and tested with experiments (13). As this cycle is repeated, the system can build and refine a knowledge base that captures some of the important features of the domain in which the experiments are being conducted.

1.2 Problem

The purpose of this thesis was to investigate methods of incorporating a machine learning component into a flight combat simulator for the purposes of improving agent behavior. Specifically, I wanted to examine the usefulness of discovery-based learning as a tool for extending the library of air combat responses available to an autonomous agent engaged in a dog-fight against another aircraft.

1.3 Research Objectives

The work accomplished as part of this investigation tested the following primary hypothesis:

Hypothesis:
Discovery-based learning can be used to improve the performance of an agent in the air combat domain.

The goal of this research was to determine whether or not discovery-based learning is suitable for a dynamic domain such as air combat. The suitability of this form of learning was assessed empirically by measuring agent performance before learning and comparing it to agent performance after learning had taken place. If the agent learned maneuvers that placed it in a better position than before, then it may be concluded that discovery was a useful tool for improving performance of the autonomous agent used during this research.

The primary investigation was decomposed into two smaller, more manageable, sub-hypotheses. The first sub-hypothesis was the *Features Hypothesis*:

Features Hypothesis:
Agent behavior can be improved by selecting a response based on a handful of key features.

The Features Hypothesis stated my belief that a reactive aircraft agent could improve behavior by using a relatively small number of parameters as an index into a table of responses, rather than always responding in the same manner. To successfully test this hypothesis, the key features were first singled out from the pool of possible features. Then, using only these features to select responses, scenarios were conducted to measure performance. Specifically, the following issues were addressed:

- During combat, what parameters are critical when selecting a course of action?
- What granularity in the critical parameters is required to improve agent behavior?
- How should the behavior of MAXIM agents be represented so that a working learning component can be integrated into the simulator?

Confirming the Features Hypothesis would contribute to the overall goal of improving agent performance, but it was completely independent of the learning aspect of this research. If the features were appropriate, then agent behavior would improve regardless of whether the responses were learned or manually entered into the system. To complete the primary investigation, another sub-hypothesis was tested:

DBL Hypothesis:
Given the division of agent responses by key features, discovery-based learning can improve behavior by testing various responses within each division and remembering those that worked best.

The Features Hypothesis proposed a possible representation for agent behavior, and the DBL Hypothesis suggested that discovery-based learning may be useful for improving combat tactics using that representation. The addition of the learning component forced me to address an additional issue:

- What domain knowledge is required for discovering air combat tactics?

If the Features and DBL Hypotheses are confirmed, then the primary hypothesis will be simultaneously affirmed.

1.4 Approach

This research built upon the MAXIM air combat simulator previously developed by AFIT students. The focus of this work was to develop and integrate a mechanism into MAXIM to select tactics, execute test scenarios, and analyze the effect of variations in the tactics. Some modifications were required to retrofit the MAXIM simulator for the learning component since agent behavior was hard-coded into the existing system. During learning, the system explored alternative responses within the MAXIM air combat world in an attempt to improve autonomous agent performance. Tactics that resulted in better agent performance during training were remembered for later application in scenarios against a more dynamic opponent.

1.5 Scope

This study was limited to a learning agent competing against a non-jinking target drone. During learning, the agent did not encounter any external threats, such as surface-to-air or air-to-air missiles, nor did the agent have any of these weapons at its disposal for

use against the drone. The agent attempted to achieve an advantage over the opponent aircraft strictly through the use of maneuvers. I believed that this would permit an accurate comparison of each of the agent's maneuvers tested during the discovery process.

The research was further limited by narrowing the range of tactics to be learned. There were a whole host of strategies that the system could attempt to improve: search, attack and evasion tactics, to name a few. I chose, however, to limit the system to learning tactics useful during the *attack* phase. Learning how to skillfully attack an adversary seemed more interesting, but it was by no means the only area in need of improvement. Clearly, there will be occasions when an agent may need to evade an adversary better skilled at attack in order to survive.

1.6 Executive Overview

During this investigation, a DBL system was successfully constructed and integrated with the MAXIM architecture. The system, referred to throughout this document as NOSTRUM, used a variation of the scientific method to explore tactics in the air combat domain. As a result of repeated experimentation, NOSTRUM was able to discover many responses that were more appropriate than the single mode of control implemented in the original MAXIM program. NOSTRUM often found responses that dramatically improved the offensive position of the agent, and it sometimes placed the agent in position for an extended shot on the target when one was not available before.

The remainder of this document presents a detailed discussion of my research, beginning in Chapter 2 with the literature review. Chapter 3 provides insight into many of the high-level system design considerations, followed by a detailed discussion of the NOSTRUM architecture in Chapter 4. Chapter 5 presents data from my experience running NOSTRUM as a learning system, as well as comparisons of agent behavior before and after learning. Finally, in Chapter 6, I present my conclusion that DBL was a useful technique in the air combat domain for improving agent behavior.

II. Literature Review

2.1 Background

The following literature review examines some of the recent research in two areas related to this work. The first section reviews several of the current approaches being used to develop realistic autonomous agent behavior. Special focus has been given to reactive, planning, and deliberative systems, followed by a short discussion on combined systems. The second section takes a look at machine learning techniques of the past, with emphasis on the differences between supervised and unsupervised learning.

2.2 Architectures for Autonomous Agents

There have been many approaches for the generation of realistic autonomous agent behavior. They span the spectrum from reactive systems, the fastest of the architectures, to completely deliberative systems which carefully calculate the best course of action using domain knowledge and sophisticated problem solving. Several of these systems are examined in this section.

2.2.1 Reactive Systems. Many behaviors that emerge during an aerial engagement are instinctive: the pilot recognizes a particular situation from training experience and reacts to counter the situation. This seemingly unconscious behavior is the basis of reactive systems, which operate using the following simple scheme:

$$\texttt{perception} \rightarrow \texttt{action}$$

Reactive systems select responses based entirely on the situation at the moment. Pre-planned actions are selected rather than generated from scratch each time the environment changes, making reactive systems less computationally intensive than other types of autonomous agent architectures. This characteristic makes reactive systems suitable for use in real-time environments where surplus time is scarce and unpredictable.

2.2.1.1 Universal Plans. Another representation of reactive behavior is possible using a *universal plan*, a concept introduced by Schoppers for use in reactive robots (26). Universal plans convey highly conditional advice in the following form:

> *If a situation satisfying condition P should ever arise while you are trying to achieve goal G, then the appropriate response is action A.*

A universal plan is organized into sets of possible situations based on the reaction desired in each situation. During execution, the system classifies the current situation and uses it as an index into the universal plan. The response assigned to that class of situations is performed, and the cycle repeats with an updated description of the environment. The behavior produced by agents executing a universal plan depends primarily on the state of the world at execution time.

An interesting feature of a system employing a universal plan is that it avoids the over-commitment problems associated with traditional planners. Agents using this method do not create a sequence of actions to achieve a future goal, but instead respond only to the current situation. Universal plan responses are geared to produce short-term gains, and a response is always available, so these systems cannot be trapped in a non-monotonic problem space. One system applying the universal plan approach is MAXIM, a prototype air combat simulator developed at the Air Force Institute of Technology (6). MAXIM was designed to produce realistic aircraft and missile agents for combat training in support of ARPA's Distributed Interactive Simulation (DIS) program.

The behavior of aircraft and missile agents in MAXIM is partitioned into various phases along with corresponding sets of actions. Missile agents operate within a single *attack* phase, but aircraft agents operate within *search*, *attack*, and *evade* phases. During simulation execution, MAXIM determines which phase of operation is appropriate for each agent using case statements: if a certain condition exists, then the phase is set to the state best suited to handle that condition. For example, when an agent detects an enemy aircraft in its vicinity it goes into *attack* phase. During *attack*, MAXIM implements a proportional control strategy to direct the agent toward the target and put the agent in position to fire on the enemy aircraft. Meanwhile, the agent continually checks for the presence of

threatening enemy missiles; if any are detected, the agent immediately transitions out of *attack* phase and into *evade* phase. While evading, the agent abandons proportionally controlled flight towards the target and instead maneuvers to escape the incoming missile.

Using a universal plan, MAXIM creates dynamic agent behavior meeting the real-time requirements of the DIS protocol. The system suffers, however, from predictability; since the universal plan is a static structure, agent behavior can be anticipated and exploited. The behaviors produced during each phase are hard-coded into the program and only take into account the most general cases. Unexpected variations in enemy behavior, such as the appearance of a missile using a proportional-derivative guidance algorithm, will not be countered well by an agent using tactics suited for a proportionally guided missile.

Pilot Decision Phases in CLIPS (PDPC) is an example of a rule-based universal plan system for intelligent air combat adversaries (9, 10). implemented in COOL, the CLIPS Object-Oriented Language, PDPC uses a rich collection of air combat domain knowledge to control agent behavior. The knowledge base contains a number of maneuvers, represented as production system rules with preconditions and postconditions, that agents may access as they try to achieve higher-level goals. The preconditions ensure that the maneuver is appropriate for the current situation, and the postconditions produce changes in the agent state, such as deltas in agent velocity, acceleration, or orientation. When the preconditions for a particular maneuver have been satisfied the rule fires, causing the aircraft to execute the selected maneuver until another more appropriate rule fires and supersedes it.

A serious drawback of PDPC is the sizable amount of domain knowledge required to implement the system. Every maneuver in the knowledge base must be explicitly stated as a set of preconditions and resultant postconditions. The developers of PDPC hand-coded approximately 200 rules specifying agent behavior. Incorporating complex domain knowledge on this scale can be an arduous process that creates an opportunity for errors to creep into the rule base during both translation and interpretation of the expert knowledge.

2.2.2 Planning Systems. The purpose of a planner is to create a suitable course of action before taking a single step (2). This is generally done by searching a state space for some sequence of operators defining a path from an initial state to a desired goal state.

Searching the entire state space is often an intractable problem, however, because the number of possible intermediate states gives rise to a combinatoric explosion. Instead, some planners search a limited number of levels in the search tree and select the plan with the best chance for success at that deepest level examined during the search. This is the technique used by Intelligent Player, a program for creating autonomous agent behavior.

Intelligent Player (IP) is a computer generated fighting helicopter that employs a chess-type look-ahead for its decision logic (12). As it generates a plan, IP constructs a decision tree of alternating agent and target response nodes. With the exception of the root node, representing the agent's current situation, each node in the tree represents a possible future position of one of the two helicopters. Nodes are computed alternately for each helicopter so that a path from the root to a terminal node defines a sequence of alternating helicopter actions, as well as the projected state of the world if those actions are taken. IP develops the tree to a tractable depth, well short of the game duration, computing heuristic scores along the way at each level. When the tree is completed, the path from the root node (identifying the current situation) to the highest scoring node at the deepest level is selected as the course of action.

Although IP was capable of generating the look-ahead tree real-time, several simplifying assumptions had to be made. Decision points for the agents were staggered so that only a single helicopter could perform an action at each branch between nodes. This is clearly not the case in actual combat, when opposing pilots will not be so polite as to take turns but will constantly maneuver to out-position the other. Helicopter actions were also discretely partitioned into a relatively small number of possible responses, significantly curtailing the potential explosion of nodes representing future world situations. In practice, there will be a range of actions possible by each helicopter. Furthermore, the look-ahead strategy will not behave well as the number of agents involved increases beyond the one-on-one scenario because of the resulting increase in the number of tree nodes.

In general, real-time planners are difficult to construct. Planning requires a great deal of information up front, is computationally intensive, and delays the arrival of suitable actions (26). Plans are also brittle in the sense that they can become instantly obsolete by external events not anticipated during plan generation.

2.2.3 Deliberative Systems. To generate an appropriate plan that can transform an initial state to a desired goal state, the planner must have access to operators that can make this transformation happen. Occasionally, however, planners are unable to find a suitable sequence of operators because of holes in the knowledge base. For example, suppose a planner has been programmed to search for a sequence of operators to satisfy the goal *get-airborne* for an aircraft agent in a starting position on the runway. Furthermore, suppose that the planner only has knowledge of the operators *extend-flaps*, *rotate*, *retract-landing-gear* and *climb*. In this scenario, the planner would be unable to find a path from the initial state to the goal because it lacks knowledge of an operator critical to the task, *engine-throttle-up*. When the knowledge available to a planner is insufficient or conflicting, a deliberative system may be able to find a solution to the problem.

The SOAR architecture is a deliberative system capable of resolving these types of impasses. More than just a generative planner, SOAR is a proposed model of human cognition capable of solving problems using first principles (25). First principles establish the basic concepts of a particular problem domain and act as building blocks from which more complex concepts can be constructed. In the previous example, a standard planner would fail because it would be unable to satisfy the goal *get-airborne* without the presence of the *engine-throttle-up* operator in the knowledge base. SOAR, on the other hand, can identify the missing knowledge and leverage first principles about flight to create a new operator to throttle the engine up. Finally, through a process called *chunking*, SOAR "remembers" the newly created knowledge in long-term memory as a solution to the impasse in case it is ever asked to get the plane airborne again.

SOAR has been used as the host architecture for the IFOR/SOAR system modeling combat behavior of autonomous aircraft agents (11). The system uses first principles of aerial combat and flight dynamics to control an agent in a one-on-one combat scenario. Behavior is modeled in IFOR/SOAR as a hierarchy of problem spaces, each allowing the agent to reason about particular types of goals. The highest-level problem space is the *mission*, and subsequent problem spaces are decomposed into finer levels of granularity, such as *intercept* and *fire-medium-range-missile*. When the agent is unable to achieve the current goal, an impasse occurs that establishes a lower-level goal of resolving that impasse.

A new problem space is created, and all the knowledge available to the agent is brought to bear as SOAR searches for a way around the impasse. Assuming that the first principles available to the system regarding flight and combat are sufficient, SOAR will eventually resolve the impasse and remember the solution for future problem solving.

Using only first principles, SOAR agents initially do not have enough knowledge to perform complex tasks without impasses occurring frequently. As impasses are resolved, more complex operators are added to long-term memory for future reference. It is interesting to note that selecting a course of action for IFOR/SOAR agents would be initially computationally intensive, but after a period of time a sufficient quantity of new operators will have been chunked such that SOAR agents actually become increasingly reactive. This raises questions about the necessity of carrying around the excess baggage of the entire SOAR problem solving architecture, considering that it would be exercised only when a reactive response wasn't already in place.

2.2.4 Combined Systems. Adhering to a single autonomous agent architecture has its disadvantages. Reactive systems are fast, but sometimes respond inappropriately or, worse yet, are unable to respond at all to unanticipated situations. Planning systems select the best course of action prior to acting, but are computationally intensive and can be brittle in an unpredictable world. Systems which are purely deliberative can also be computationally intensive, but they can generate solutions to unforeseen problems. Combined, these factors are the impetus for hybrid systems which integrate reactive, planning, and deliberative components into a single architecture.

One such combined system is RPD, an architecture that integrates reactive, planning and deliberative system components for a multi-agent environment (1). The reactive component of RPD accommodates routine situations that the system can deal with on an "instinctive" level. When non-routine situations are identified, RPD passes the task to a planner that can employ both generative planning and case-based methods to construct a plan. Anytime during this process, however, RPD may not be able to disambiguate the situation or find a suitable plan to deal with the situation. In these instances, a decision making module is called to resolve the problem.

In a sense, the IFOR/SOAR architecture can also be considered a combined system, although the boundaries between the individual components are not as clearly defined as they are within the RPD system. IFOR/SOAR begins as a completely deliberative system as it solves higher-level goals with low-level first principles, but eventually evolves into a reactive system as new operators are chunked. In time, an IFOR/SOAR agent will familiarize itself with many situations and will have a corresponding action already present in long-term memory for each. When it encounters unfamiliar territory, IFOR/SOAR can fall back upon its sophisticated problem solving architecture to find a solution to new problems.

2.3 Machine Learning

No matter which architecture is chosen to implement an autonomous agent, each encounters the difficult problem of knowledge acquisition, a major bottleneck in the construction of many AI systems (24). Typically, domain knowledge is gathered by a knowledge engineer. The knowledge engineer must determine the types of knowledge needed by the system to perform the task, as well as the level of detail required. Once these have been decided, an expert is usually consulted to identify and explain all of the information that is relevant to the problem solving that will be performed by the system. Finally, the knowledge engineer must translate the knowledge into a usable form and enter it manually into the system.

Machine learning is an AI technique that can be a useful in circumventing the knowledge acquisition bottleneck by allowing the system to build a domain knowledge base through experience. The remainder of this literature review will focus on the topic of machine learning, with emphasis on those types that appear suited for learning tactics in the air combat domain.

2.3.1 Learning Perspectives. There are many viewpoints in AI literature enumerating the purposes and benefits of learning. While these definitions are directed at learning in general, they have had a direct impact on the learning mechanisms integrated with AI systems that model the human problem solving process. Most definitions agree that learning is the acquisition of experiential knowledge. The definitions diverge, however,

when examined at the following levels: what is the expected payoff of learning, and how must new knowledge be represented?

Simon defines machine learning as changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and effectively the next time (29). Adaptation of the system is possible through the refinement of existing knowledge or the addition of new knowledge. As improvements are made to the knowledge base, system performance should improve and become more robust. In many ways, defining learning as performance improvement is similar to another view that learning is the process of skill acquisition. This definition clearly distinguishes having the knowledge to perform a task from being able to perform that task well. The skill acquisition perspective draws heavily upon recent research in cognitive psychology, where the effects of the power-law of practice have been identified (25).

On the other hand, Dietterich states that learning can be defined simply as an increase in knowledge (5), regardless of the impact the new knowledge has on system performance. His definition of learning includes not only facts explicitly known to the system but implicit facts as well, using the following axiom of knowledge closure:

> *If an agent knows a body of facts, F, then the agent also knows any facts that are deductive consequences of F. In other words, an agent knows the deductive closure of his knowledge.* (4)

Since the goal of this research is to produce more robust autonomous aircraft agents, capable of improving performance with experience, Simon's definition of machine learning is more appropriate and will be implied throughout the remainder of this review unless stated otherwise.

2.3.2 Learning Types. Research in machine learning can, in general, be separated into two classes: systems that require the assistance of a human teacher to learn new concepts (*supervised learning*), and systems that can learn new concepts without any teacher involvement (*unsupervised learning*). Supervised learning requires a teacher to collect and present positive and negative examples of a concept. From the examples, the system is able to identify the important attributes of a concept while filtering out those that are

irrelevant. Unsupervised systems, however, can acquire knowledge by modeling the process of scientific experimentation, learning from the results produced when a hypothesis is tested. Because experiments can be conducted in a simulated world, little or no teacher involvement is required. Within these two categories there are a variety of techniques that can be leveraged, so a closer examination is warranted.

2.3.2.1 Supervised Learning. Rote learning is the most basic of the machine learning methods (24). In its simplest form, rote learning is nothing more than knowledge manually programmed into the system or entered into a database. This definition is rather trivial because it encompasses virtually every keystroke of program code or data that can be entered into a computer. A more useful form of rote learning was employed by Samuel's checker program (2). In that system, rote learning was used to memorize checker board positions and the point value assigned to each board by an evaluation function. The program recalled previously memorized boards during later play when look-ahead search was used to select the next move. Using this approach, Samuel's program was able to look deeper into the search tree than would have otherwise been possible in a reasonable period of time, improving its performance with experience.

More commonly, supervised learning systems use induction to reduce the burden on the human instructor and to speedup the learning process. Inductive systems attempt to learn general concepts from a set of training examples. Mitchell introduced version spaces as an inductive method that learn concepts from positive and negative examples presented by a teacher (27) Initially, the version space is as large as the entire concept space. As concepts are presented, the system updates a set of general and specific descriptions of the concept. The general and specific sets converge to define a narrower version space as additional examples are presented to the system. The advantage of inductive learning methods is that the teacher does not get bogged down in the problem of knowledge representation since the system makes the necessary transformations, although the teacher must select and order the training examples.

Explanation-based learning (EBL) is a deductive form of learning that can reduce some of the burden normally placed on a teacher using an inductive system. EBL systems

can generalize a concept through the examination of a single positive example. The system relies heavily on domain knowledge to identify the relevant attributes of a concept. To begin learning, an explanation-based system is presented with four inputs (3):

- **A training example:** A positive example demonstrating the concept to be learned.
- **A goal concept:** A high-level description of what the program is supposed to learn.
- **Operationality criteria:** Specifications for concept definition that will allow efficient concept recognition in the future.
- **Domain theory:** Domain specific knowledge that can be applied towards the generalization.

Once presented with these inputs, the explanation-based learning system eliminates the unimportant aspects of the training example and generalizes those that appear to be relevant. The advantage of EBL is that only a single example is required to generalize the important characteristics of a concept, unlike most inductive systems which require several training examples. However, the domain knowledge used by EBL must be accurate and complete for a proper generalization to be made.

The chunking mechanism built into SOAR is very much akin to the type of learning performed by an EBL system. When SOAR discovers an impasse during problem solving, it reaches into long-term memory for every piece of domain knowledge that may help resolve the impasse. When a solution to the impasse is found at this lower level, SOAR generalizes as much of the solution as possible so that knowledge learned during the resolution of a single impasse might be put to use during similar, but not exactly identical, situations (16). In this way, a single problem-solving episode serves as the training example from which future problem-solving solutions are created and remembered.

2.3.2.2 Unsupervised Learning. Unsupervised learning systems offer an exciting alternative to the concept collection and presentation required by the inductive and deductive learning systems discussed in the previous section. One form of unsupervised learning, known as *discovery*, has been developed to model the process of the scientific

method: "constrain attention to a manageable domain, gather data, perceive regularly in it, formulate hypotheses, conduct experiments to test them, then use the results as new data with which to begin the cycle again" (21). Two such discovery system, BACON and AM, will be examined in this section.

BACON was a system that used empirical methods for discovering numeric laws. Six different versions of BACON, each of increased sophistication than its predecessor, used a variety of heuristics and discovery techniques to find a mathematical function that described the relationship between two numeric terms. By examining pairs of data, BACON was able to discover Kepler's third law of planetary motion, Galileo's laws for the pendulum and constant acceleration, and Ohm's law (18).

Initial versions of BACON used four simple heuristics to uncover useful relationships in observed two-variable data. The first two heuristics were used by BACON to conjecture that if a particular relationship was identified in a number of cases then it could well be true for all cases. The remaining heuristics were used to recursively create additional numeric terms from the product and ratio other pairs of terms. Guided by the heuristics, BACON worked its way through the search space until the application of the product or ratio operator produced a constant term throughout the data set; at that point, the correct relationship had been identified. The practice used by BACON to create new terms from existing ones is reminiscent of constructive induction learning. This technique can be used when the data presented to the system contain no directly relevant descriptors, but combining the data in some fashion had potential for producing something more useful to the learning task (23).

For this discussion pertaining to BACON, Dietterich's definition of learning as defined in section 2.3.1 may be more appropriate. If the system is not viewed from this perspective, it is difficult to identify what learning is actually being accomplished by BACON. The inductive process used to extract a mathematical equation from empirical data can be equated to making implicit knowledge explicit. Although Dietterich's definition specifically refers to the deductive closure of knowledge, Simon's definition of machine learning is certainly not adequate since the system is not refining its ability to perform a task.

Another approach, known as discovery-based learning (DBL), has been used to demonstrate knowledge acquisition by performing and analyzing experiments. One of the most well known DBL systems was Lenat's AM, a program for the discovery of new mathematical concepts and the relationships between concepts (19). By performing experiments in a mathematical world, AM discovered scores of well known concepts, such as addition, multiplication, prime numbers, and DeMorgan's theorems.

Unlike explanation-based learning, there is no domain knowledge requirement in a DBL system. Instead, DBL systems rely upon a set of heuristics. Heuristics give the system a way to represent rules-of-thumb that produce positive results most of the time, but are not always guaranteed to work. For example, a heuristic used by AM during its discovery process was "*if a function is found to produce an interesting result, then look at its inverse.*" Once AM had discovered the concept of multiplication, this heuristic lead it quickly to the concept of division.

The experimentation process was controlled in AM through an *agenda*. The agenda rank-ordered all of the tasks to be carried out based on an interestingness value assigned to each task. The heuristics built into AM were used to determine which experiments were more interesting than others, and to make suggestions about future experimentation. At the start of each cycle, AM would remove the first task from the agenda and conduct an experiment to carry out that task. Based on how interesting AM determined the results to be, new tasks might be added to the agenda, and the cycle would repeat.

DBL was also used by a researcher at AFIT to develop threat avoidance maneuvers in the MAVERICK learning system. MAVERICK demonstrated the ability to autonomously create a set of robust maneuver templates for reactive route re-planning around unforeseen ground-based threats (13). Through repeated experimentation, MAVERICK developed maneuver sequences to safely traverse an area populated with surface-to-air missile sites. Each maneuver sequence was scored based on the trip time across the area and positive radar contact time. Maneuvers that minimized both were considered more interesting and, consequently, generated new responses that were explored first.

By the very nature of their operation, DBL systems require a world in which to perform experimentation. Experiments could be accomplished in the real world, with humans or robots performing the tasks and reporting back the results. Another approach is to allow a DBL system to experiment in a computer simulated world. This method allows experimentation to proceed at the processing speed of the computer, instead of the much slower rate to be expected from human experimenters. It is important to realize that the scope and quality of knowledge learned by a DBL system conducting experiments in a simulated world is limited by the fidelity of the simulator.

2.4 Conclusions

This chapter provided an overview of some of the research related to this thesis from two areas, autonomous agent architectures and machine learning.

A variety of autonomous agent architectures have been tried. Reactive systems are suitable for real-time applications because they are fast, but they are sometimes limited in the range of behaviors they can produce. Planners in a real-time environment can be troublesome because it takes time and computing power to produce a good plan, and even a good plan is brittle when unforeseen events occur. Deliberative systems, such as SOAR, offer much potential as models of human cognition but may also require more time and computing power than is available in a real-time environment. Combining components from each type of architecture is an attempt to make use of the best features of each component without suffering the overall disadvantage of a single architecture.

The field of machine learning has developed a great deal since rote learning was first implemented in Samuel's checker program. Other supervised learning techniques, such as concept induction and explanation-based learning, are less labor intensive on the part of a human teacher than rote learning but still require the assistance of a teacher to collect and present concept examples. An explanation-based learning system is able to learn from a single positive example, but draws heavily on its domain knowledge to identify attributes of the example that match the goal concept. BACON, an unsupervised learning system, explored a data set for useful relationships without teacher assistance by modeling the

process of scientific discovery. Likewise, AM can explore a simulated world in much the same way as it uses heuristic search to discover useful concept in that world.

NOSTRUM, the system produced as a result of this research, borrowed heavily from concepts employed by the discovery systems AM and MAVERICK to improve the universal plan responses of air combat agents. A high-level discussion of the NOSTRUM architecture follows in Chapter 3.

III. Methodology

3.1 Overview

The previous chapter summarized some of the current research thrusts for generating realistic autonomous agent behavior. MAXIM, a prototypical system for autonomous aircraft agents, uses a reactive planning strategy to simulate behavior during air combat that is often predictable and unrealistic. For this thesis, I chose to investigate a strategy for integrating machine learning into MAXIM to extend its range of behaviors with the intent of producing more robust behavior.

This chapter describes the considerations made during the design process of NOSTRUM, the system developed as part of this work for enlarging MAXIM's universal plan[1]. The finer details of NOSTRUM's architecture will be addressed in Chapter 4; this chapter is dedicated to a higher-level discussion for the purpose of answering the following questions:

- What tactics could be learned that would be the most interesting and have the greatest impact on improving overall agent behavior?
- Of the various machine learning methods, which was best suited for learning in the chosen domain?
- Which language or platform was the best for implementing NOSTRUM?
- What factors identify the key parameters used by a pilot to select different tactics during an engagement? What information does a pilot use to select one maneuver over another?
- Given the domain, the structure of MAXIM, and the method of learning chosen, what approach could be taken to expand the universal plan?
- What is the basic structure of a system capable of learning air combat tactics?

[1] Henceforth, references to MAXIM refer to the *simulator* portion of the system, whereas NOSTRUM is the *learning* component of the system.

3.2 Domain

Since MAXIM is an air combat simulator, the domain chosen for this research was confined to the air combat world. Within the world created by MAXIM, however, there were several types of tactics that a machine learning system could explore.

As mentioned in Section 2.2.1.1, the current version of MAXIM operates in one of three phases and performs actions which are applicable to that phase. Depending on the presence of a target aircraft or threatening missile, an agent's phase will be *search*, *attack*, or *evade* (6). MAXIM agents always fly directly towards the target aircraft using proportional control, firing a missile at the target as soon as minimum range and track angle constraints have been met, While this type of control is sometimes used by pilots, there are many other behaviors that are appropriate in other situations. A pilot may launch a missile at a target as soon as the aircraft is within range, but might then maneuver for position behind in case the missile doesn't destroy the target. An agent that relies solely on proportional control would not always be able to achieve a positional advantage, and would rarely be victorious against a skilled opponent.

The behavior for each of the three MAXIM phases was hard-coded into the program when this research began. The code was therefore limited in the range of behaviors that it could produce, making each phase a candidate for expansion through machine learning. I chose to concentrate my effort towards improving agent behavior in the *attack* phase because it seemed the most interesting and could have an impact on improving the overall agent behavior: a skilled agent that can acquire and destroy a target better than its opponent may not rely on evasion tactics as frequently as an unskilled agent.

3.3 Machine Learning Method

Of all the machine learning methods that could be leveraged by a system to improve agent behavior, discovery-based learning stood out as the most appealing technique. My personal experience in the air combat domain was limited to skill acquired during play with toy flight simulators, and there was precious little expert knowledge readily available. Other autonomous agent systems, such as the IFOR/SOAR project that uses an explanation-

based learning approach to chunk solutions as impasses are encountered, or the rule-based approach of PDPC, require a substantial amount of domain knowledge to be hand-coded into the program before reasonable agent behavior is possible.

DBL systems, on the other hand, do not require human interaction and learning can begin with very little domain knowledge. Instead, a DBL system uses a set of heuristics to propose, select, and perform experiments in a simulated world as the system attempts to learn characteristics of that world. As mentioned in Section 2.3.2.2, a researcher at AFIT successfully used DBL in the MAVERICK system to develop maneuvers around ground-based missile threats. The knowledge built into MAVERICK was limited, requiring only fourteen heuristics tuned to the route-planning domain. Previous success with DBL in the MAVERICK system, combined with the fact that MAXIM provides an interesting dynamic world in which to experiment with minimal requisite domain knowledge, motivated me to select DBL as the learning mechanism for enlarging MAXIM's universal plan.

3.4 Implementation

MAXIM is written in CLOS, the Common Lisp Object System. Common Lisp is an interpreted language, making it a relatively flexible environment that eliminates the time-consuming compilation step during development. Lisp implementations that adhere to the ANSI standard have an integrated compiler as well, so optimized run-time binaries can be created after program development. Given the inherent flexibility of Common Lisp, combined with the fact that it would be easier to integrate code controlling the discovery process in the language native to MAXIM, CLOS was chosen as NOSTRUM's implementation language.

Early in my research, I considered using SOAR as the host architecture for the discovery portion of the system. SOAR is a proposed general model of human cognition that attempts to satisfy goals using the *problem space hypothesis.* This hypothesis states that all problem solving can be accomplished by selecting a sequence of operators that defines a path from some initial state to a goal state (15). There is a great deal of similarity in the problem solving method implemented in SOAR and models of the human process of discovery (17). The experimentation process can be seen as repeated problem solving within

problem spaces for generating, selecting, conducting, and assessing the outcome of experiments. Problem solving within some top-level problem space, such as *experiment-world*, could repeat until a desired experimental outcome has been reached.

Connecting SOAR to the world provided by MAXIM, however, would not have been a trivial matter. Passing data back and forth between SOAR and an external program is done through working-memory augmentations to the state (15), which proved with further investigation to be too clumsy for the amount of communication necessary between MAXIM and the DBL system. There was little to be gained from the notion of integrating SOAR into the MAXIM architecture, other than the novelty of using SOAR for this purpose.

3.5 Factors Affecting Plan Selection

To make MAXIM a more effective opponent during the *attack* phase, it was necessary to examine the domain and identify the parameters which are important to a pilot during combat. A skilled pilot will be able to take into account many different parameters before choosing his next response, but there might only be a handful of especially critical variables. Reducing the number of these parameters would be helpful during the learning phase, at which time the system must determine what effect each experiment has had on their resultant values.

In the case of MAXIM, I decided that an improvement in performance might be achieved if responses were selected from the universal plan based on the following five parameters: angle between fighter's line-of-sight and the the target's direction of travel (aspect angle), fighter to target separation (range), relative altitude, relative velocity, and relative heading. As shown in Figure 3.1, every 30 degree increment in aspect angle defines a *plan sector* for which there can be several possible universal plan responses. Within sectors, plan selection was further broken down based on range to the target (range1 or range2), relative altitude from the target (above, nearly equal, or below), velocity relative to the target (faster or slower), and approximate heading relative to the target (parallel or opposing), for a grand total of 288 plan responses. After examining flight training documents and discussing the problem with an F-4 pilot, I believed that a universal plan

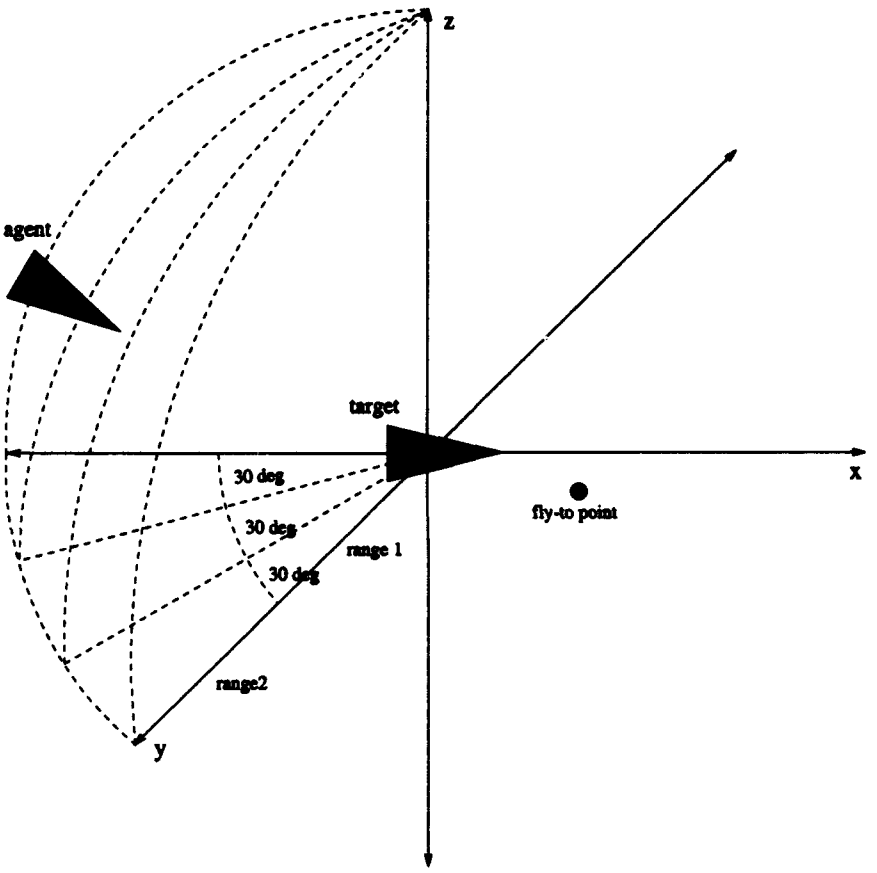

Figure 3.1 **Plan sector divisions.** The attack response chosen by an agent depends primarily on which plan sector the agent is in. Responses are further refined on the basis of relative velocity, range, relative altitude, and relative heading.

divided into segments by these features, and at this resolution, might be less predictable and more robust than one that used the same tactic under all circumstances (31, 22).

3.6 Learning Approach

To extend MAXIM's universal plan during the attack phase, I decided to train an agent in a 1-v-1 combat scenario against a level-flying, non-jinking bogey moving at constant velocity. MAXIM agents are normally quite aggressive, hurtling straight toward an opponent and firing a missile as soon as minimum range constraints are met. During learning, however, the ability to fire these all-aspect missiles was disabled. The learning goal of the agent was to achieve a positional advantage over the target, placing itself at the proper range and track angle for a missile shot at the target. Once learning had been completed, the all-aspect weapons capability was restored, but training with this capability off gave NOSTRUM an opportunity to learn contingency plans. My hope was that a response learned during a specific training scenario against a non-jinking target would be applicable against dynamic targets as well.

NOSTRUM was set up to test possible responses for variations in aspect angle, relative velocity, range, relative altitude, and relative headings. The number of plan sectors to explore was reduced by using the symmetry of fighter-target geometry to mirror the responses on the right- and left-hand sides of the sphere surrounding the target aircraft. This cut the number of unique responses to be learned in half; responses needed when the agent approached from the other side of the target were mirrored by changing the sign of appropriate terms in the corresponding known response.

Before learning began, each of the 144 plan responses was initialized so that MAXIM would fly directly towards the target. This was done so that the default behavior of NOSTRUM was at least as good as that of a standard MAXIM agent. During learning, however, scenarios were executed so that NOSTRUM could experiment with alternatives for each response. Alternative responses were suggestions for repositioning the *fly-to* point of the agent which, in the case of proportional control, was exactly equal to the current location of the target. Responses could also be modified by suggesting changes in the magnitude of the agent's acceleration vector. By moving the fly-to point to some other position relative

to the target, and adjusting aircraft power settings, I believed that NOSTRUM would learn other types of pursuit such as *lead* and *lag*, and when these types of behaviors were appropriate. In addition, I expected that the agent would learn energy management maneuvers by selecting fly-to points that were above and below the target aircraft. Fly-to points are discussed in greater detail in Section 4.2.2.

3.7 System Architecture

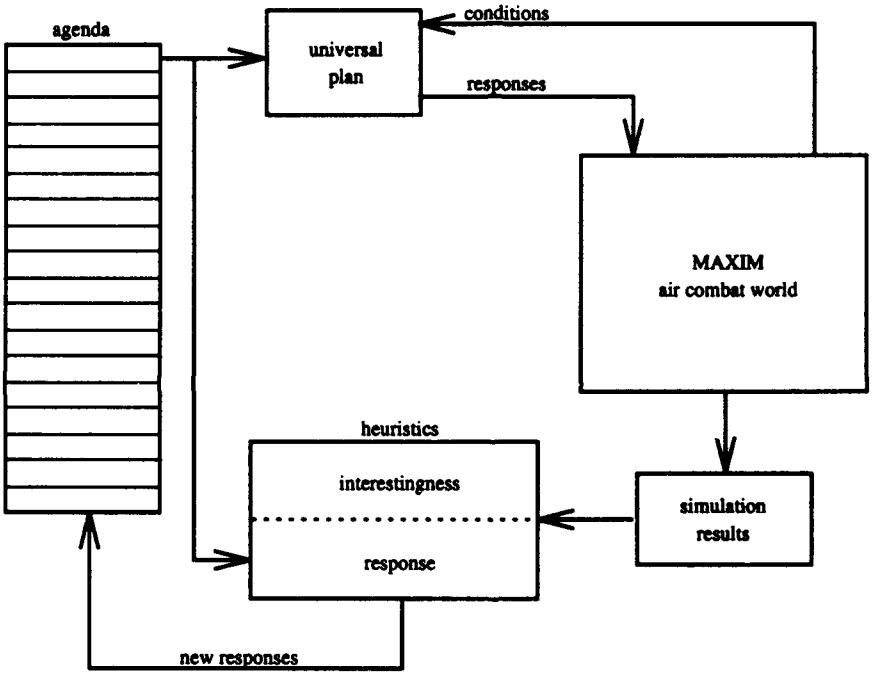

Figure 3.2 NOSTRUM **block diagram.** An architecture for discovering air combat maneuvers in MAXIM.

Figure 3.2 illustrates the architecture developed for NOSTRUM, which resembles the design of both AM and MAVERICK. An important structure in a DBL system is the *agenda*, a priority queue of experiments to try next, rank ordered according to their *interestingness*. Interestingness is simply a measure of a particular experiment's worth based on a set of scoring heuristics. Experiments that have received the highest scores spawn additional

experiments that are explored first. This allows experimentation to continue along paths that appear to have the highest probability for success.

The success or failure of responses was measured using heuristics appropriate for the flight combat domain. Since the desired goal was to let NOSTRUM find ways to achieve an advantageous end-game position, maximizing shot time at the target while reducing the target's shot time on the agent, two sets of the heuristics were required to guide experimentation along these lines. The first set of heuristics, known as *interestingness heuristics*, were used to assess the quality of a response following a simulation run. Interestingness was calculated using not only by the quality of the response itself, but by predicting improvements that might result if the system explored modifications to that response. Predicting progress towards a better solution was attempted by looking for improvements in a better end-game position, even if the change did not immediately result in an increase in shot time or a reduction in danger time.

A second set of heuristics known as *response heuristics* was then used to suggest transformations to each response explored by the system. The MAVERICK system used a technique similar to this to improve the flight time and decrease radar coverage of an aircraft flying through an area populated with SAM sites. NOSTRUM's response heuristics could suggest adjustments to the fly-to point relative to the target aircraft, or suggest an increase or decrease in the power setting. An indefinite number of response heuristics could trigger during the generation phase, suggesting zero or several new *child* responses to try next. The list of child responses was inserted into the queue of pending responses, with the parent that produced them, at a position fixed by the interestingness of the parent. The system then selected what appeared to be the most promising response from the queue, and the cycle repeated.

Theoretically, experimentation could continue indefinitely, but for practicality learning had to be stopped after some number of responses had been explored. At that point, the best response tested so far became the response for that portion of the universal plan. The system was reset and learning began for the next universal plan response.

3.8 Summary

This chapter presented a high-level discussion of the approach chosen for this thesis. Effort was concentrated on improving tactics used by a MAXIM agent during the *attack* phase because it seemed to be the most interesting and had promise for improving overall agent behavior. While in the attack phase, the MAXIM program was altered to select the most appropriate response for the autonomous agent based on five key features that capture important aspects of the engagement. Manually determining what is an "appropriate" response for each of the 144 distinct plan sectors would have been a laborious project. The potential speed up in knowledge acquisition offered by the integration of a machine learning technique is an exciting prospect.

There are many flavors of machine learning, but discovery-based learning was selected as the method of choice for two reasons. First, DBL systems typically have only a small domain knowledge requirement to begin learning. This was appealing because I was not an expert in the air combat domain. Second, discovery-based learning is a form of unsupervised learning. By conducting experiments within each plan sector, the system can automatically modify and test a number of responses, and remember the one that appears to work best.

IV. NOSTRUM *Design and Implementation*

4.1 Overview

Two programs, MAXIM and NOSTRUM, were integrated to produce the learning system constructed during this research. MAXIM created a simulated world for competing autonomous aircraft agents, and NOSTRUM was plugged in to this world so it could test and measure the performance of variations in air combat tactics. This chapter highlights many of the detailed design decisions made while integrating the two systems. Also included is an in-depth discussion of NOSTRUM's operational cycle and the expected system behavior.

4.2 MAXIM

MAXIM is a simulated air combat system developed by students at the Air Force Institute of Technology. Originally developed as a class project in support of ARPA's Distributed Interactive Simulation program, MAXIM has also been used to demonstrate the initial success of purely reactive agent behavior in the air combat domain (7). A serious criticism of reactive planning in general is that it is impossible to anticipate and include the most appropriate plan for every situation the agent might encounter. Therefore, there will be situations where the reactive response falls short and the resultant behavior exposes deficiencies in the universal plan.

Being a reactive system, this criticism applies to MAXIM as well. The behavior of a MAXIM agent is hard-coded into the system and plan response selection is based on the presence of external threats such as enemy aircraft and missiles. The first mode of agent operation is *search*, during which the agent flies straight and level for the most part, occasionally turning right as it scans for enemy aircraft. Once an enemy is identified, the agent immediately transitions to *attack* mode. During attack, the agent relies exclusively on a proportional control strategy to engage and, ideally, destroy the target with a missile. If the agent is threatened by a missile launched by the enemy aircraft, the agent abandons attack for the moment and switches to *evade* mode. While evading, the agent flies a course perpendicular to the direction of missile flight as it tries to turn inside the missile moving

at a much greater velocity. If the evasion is successful, the agent then resumes its attack on the nearest enemy aircraft.

MAXIM agent behavior during search, while rather monotonous, is not as critical to the survival of an actual pilot as are the attack and evade strategies. The availability of on-board radar systems and AWACS simplifies and accelerates the process of identifying enemy aircraft, reducing the requirement for this type of behavior in the cockpit. In practice, however, MAXIM agent behavior during the attack and evade phases is inadequate because simplifying assumptions were made during program development. For example, the proportional control strategy used by MAXIM during the attack phase will work well when the age is directly behind the enemy aircraft and traveling at the same velocity, but it is not as appropriate when the agent is traveling faster than the enemy; using only proportional control, the agent could overshoot the enemy and make itself vulnerable by giving the enemy a missile launch opportunity. A different response, directing the agent to prevent an overshoot by bleeding off airspeed through a climb, might work better in this scenario.

4.2.1 Architecture. MAXIM makes extensive use of the object-oriented features of its implementation language, CLOS. There are object classes defined for aircraft and missile agents, and additional classes define simulation and Ethernet manager objects. MAXIM works quite nicely as a stand-alone air combat simulator, but the Ethernet manager object gives MAXIM an interface to the outside world for combat against other simulators.

Simulations are controlled in MAXIM by alternating friendly and enemy object updates. Each "side" of the simulation has its own simulation manager that maintains a list of all objects known to be participating in the simulation. During the first half of the update cycle, the manager requests updated enemy object information and inserts each packet received in a history list. When all new information has been received, the simulation manager updates each friendly object by sending it an update message tagged with the current simulation time. During the update, friendly objects select an appropriate phase and, if appropriate, a target aircraft selected from the history list to fly towards. Agents then use the delta in time since the previous update to compute new position, velocity,

acceleration, and orientation information, which is then passed to the Ethernet manager for network transmission.

4.2.2 Fly-to Points. Agent control in MAXIM is implemented through the selection of an appropriate *fly-to* point. The fly-to point is simply a three-dimensional identification of a point in space that the agent should attempt to reach. For flying straight and level, as an agent does in the search phase, the fly-to point is always directly in front of the agent; during the attack phase, as shown in Figure 4.1, the fly-to point becomes the coordinates of the enemy aircraft; when evading, MAXIM computes a fly-to point that is perpendicular to the missile velocity vector. Once selected, MAXIM determines the maneuvers necessary to move the agent in the direction of the fly-to point and implements them.

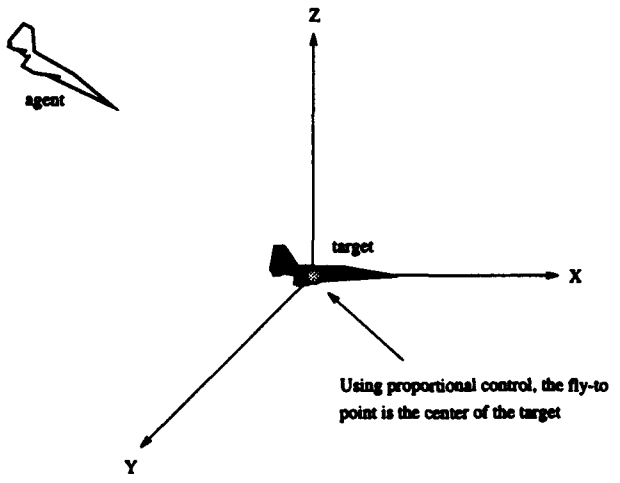

Figure 4.1 **The proportional fly-to strategy used by standard agents.**

Using fly-to points to control agent maneuvering can result in extremely flexible behavior. Adding an offset to the target position in the direction of target flight will force a MAXIM agent into lead pursuit, while adding the same offset in the opposite direction causes a lag pursuit. Further adjustments can be made to the fly-to point by adding offsets

above and below as well as to the left and right of the target aircraft, as shown in Figure 4.2. In effect, the target aircraft position becomes the origin of a local coordinate system rotated in the direction of the target aircraft's velocity vector. Offsets from the target nose, wing, and tail specify a local (X Y Z) coordinate, which is then rotated onto the fixed world coordinate system to produce the actual fly-to point.

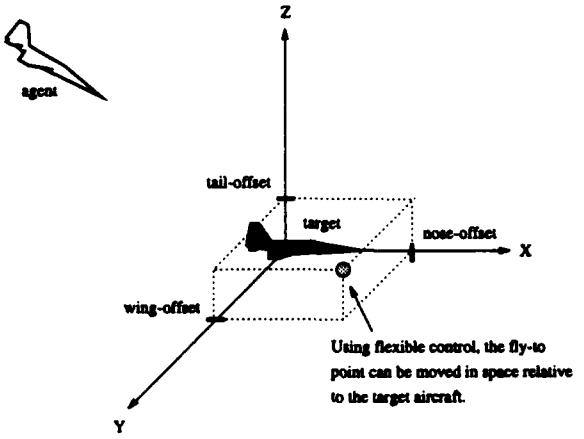

Figure 4.2 **The adjustable fly-to point strategy used by flexible agents.**

An additional feature was incorporated into MAXIM to support several levels of acceleration and deceleration in the form of a *power setting.* Before this enhancement, MAXIM agents either accelerated or decelerated at the maximum possible rate, toggling between full after-burner and engines fully cut-off. Pilots seldom operate their aircraft in this manner. The power setting allowed gradations between full acceleration and deceleration in the form of a percentage to be applied.

4.3 MAXIM *Modifications*

During program development, I decided to retain the majority of the original MAXIM architecture and modify only those parts of the code necessary to support the NOSTRUM discovery process and testing. This section describes some of the more significant changes that had to be made while integrating the two systems.

4.3.1 Agent Types. Rather than strip functionality from the original MAXIM program I thought it would be best to instead add new functionality. This approach made it possible to compare and contrast the behavior of a standard MAXIM agent with that of a learning NOSTRUM agent. The first step was to create *standard* agents that exhibited the default proportional control behavior common to all MAXIM agents and augment the program with *flexible* agents. Flexible agents select adjustments to the fly-to point and power setting from a table learned through discovery, maintained internally as a five-dimensional array.

4.3.2 Dealing With Nondeterminism. I initially considered using standard MAXIM agents as the target aircraft while experimenting with the attack responses of the flexible agent but decided against it because of the non-deterministic characteristic of MAXIM: running the simulation multiple times with identical inputs often produces different results. MAXIM behaves differently from run to run for two reasons. First, the program uses a real time clock to determine time deltas between agent updates, and it is highly unlikely that the deltas will be identical because of variations in system loading. Second, agents enter into the program in search mode and randomly turn right with a probability of 79% during the initial moments of a simulation. During one test run, an ill-fated right turn made at the beginning of a simulation put the agent at a disadvantage and finally resulted in its destruction, when it would have otherwise emerged unscathed.

Measuring progress during discovery is simplified when the experimental results can be attributed directly to the hypothesis being tested and not to external influences that cannot be controlled. I attempted to minimize the nondeterministic quality of MAXIM by first creating a third type of agent, specifically for use during discovery, known as the *drone*. Drones fly straight and level at a constant velocity from an initial position at a fixed heading, thereby preventing a random right turn during the first few moments of the simulation. Since its behavior never changed, the drone also provided a stable platform against which competing responses could be compared. Nondeterminism was further minimized by changing the way the simulation dealt with time. Instead of using unpredictable deltas from a real-time clock, I added the ability to switch to *pseudo-time*. When using

pseudo-time, the program automatically increments the time base by a fixed amount so that random fluctuations in time deltas between updates is no longer a possibility.

The drone agent type was not permitted to make a random turn during search, but there was still a possibility that the agent being tested might make such a turn. To prevent this from happening, a third MAXIM modification was added to disallow random turns during search when learning was enabled. I made the assumption that the agent would already "know" that it was engaged against an enemy aircraft, and therefore searching for the target would be unnecessary.

Together, these three modifications completely removed nondeterminism from the simulation. The net effect was that response performance was more accurately compared against other responses and not against unpredictable aircraft behavior.

4.3.3 Universal Plan Representation. The proportional control behavior of standard agents was hard-coded into the original MAXIM program. Flexible agents, however, required a look-up table so that the most appropriate attack response could be chosen real-time based on the current situation. A multi-dimensional array was used to represent the table, and each element in the array was indexed by the five parameters used to assess the current situation:

- **Relative velocity** The difference in the agent and target velocities: if positive, the agent is moving faster than the target. The agent's relative velocity was represented as being either *faster* than or *slower* than the target, for a total of two possible values.
- **Heading crossing angle** Represented the angle between the target and agent velocity vectors: if less than 90 degrees, a portion of the agent's velocity was in the same direction as the target's velocity vector. The heading crossing angle was represented as being either *parallel* or *opposing*, for a total of two possible values.
- **Range** The magnitude of the line of sight vector from the agent to the target. Range was represented as being either *close-in* or *far-away*, for a total of two possible values.

- **Relative altitude** The difference in the agent and target altitudes: if positive, the agent is above the target. Altitude was represented as being *above*, *nearly-equal*, or *below*, for a total of three possible values.
- **Aspect angle** The angle measured from the target aircraft tail to the agent. The aspect angle was represented in 30 degree increments surrounding the target, for a total of twelve possible values.

Figures 4.3 through 4.6 illustrate graphically how four of the five key parameters were used to create a sphere of plan sectors surrounding the target aircraft. The fifth parameter, difference in agent and target velocities, is not shown because it is a measurement used to indicate if the agent is moving faster or slower than the target, and is computed simply by subtracting target velocity magnitude from agent velocity magnitude.

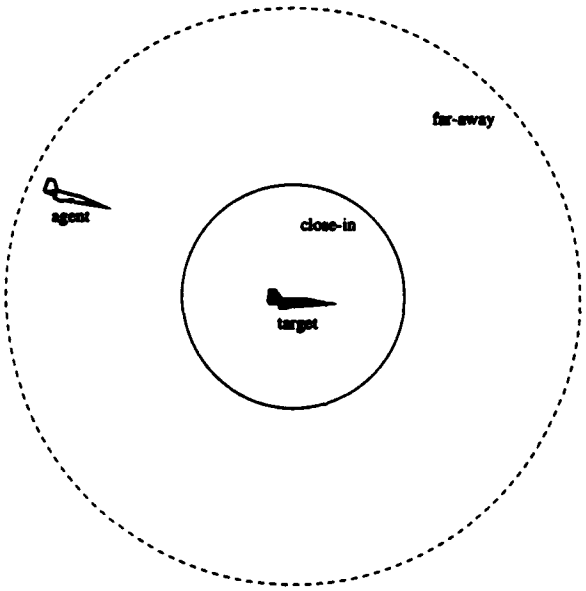

Figure 4.3 **Plan sector range divisions.** Two concentric spheres surround the target to select responses when the agent is close to or far away from the target. The outer range boundary shown does not actually exist: anything that is not *close-in* is *far-away* by default.

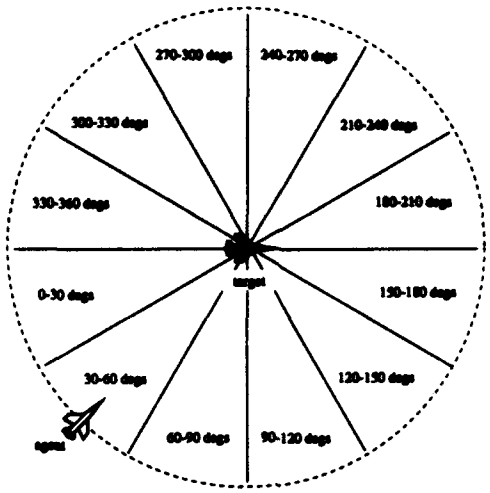

Figure 4.4 **Plan sector aspect angle divisions.** Viewed from the top, the sphere surrounding the target is divided into twelve equal slices of 30 degrees each. NOSTRUM learns the responses in on the right side of the target, and mirrors these responses when the agent is on the left side.

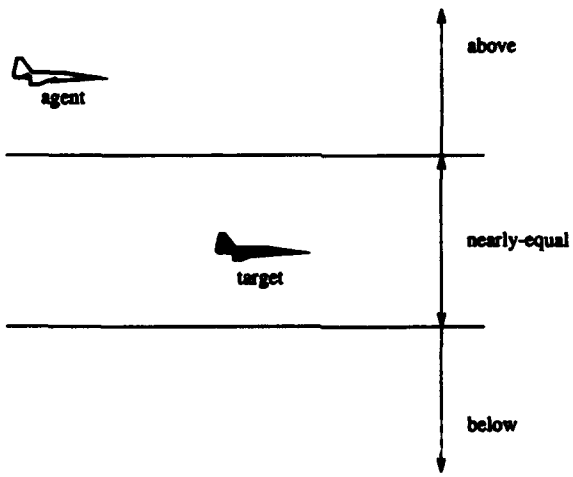

Figure 4.5 **Plan sector altitude divisions.** Viewed from the side, there are three altitude divisions identifying when the agent is above, below, or at nearly equal altitude as the target.

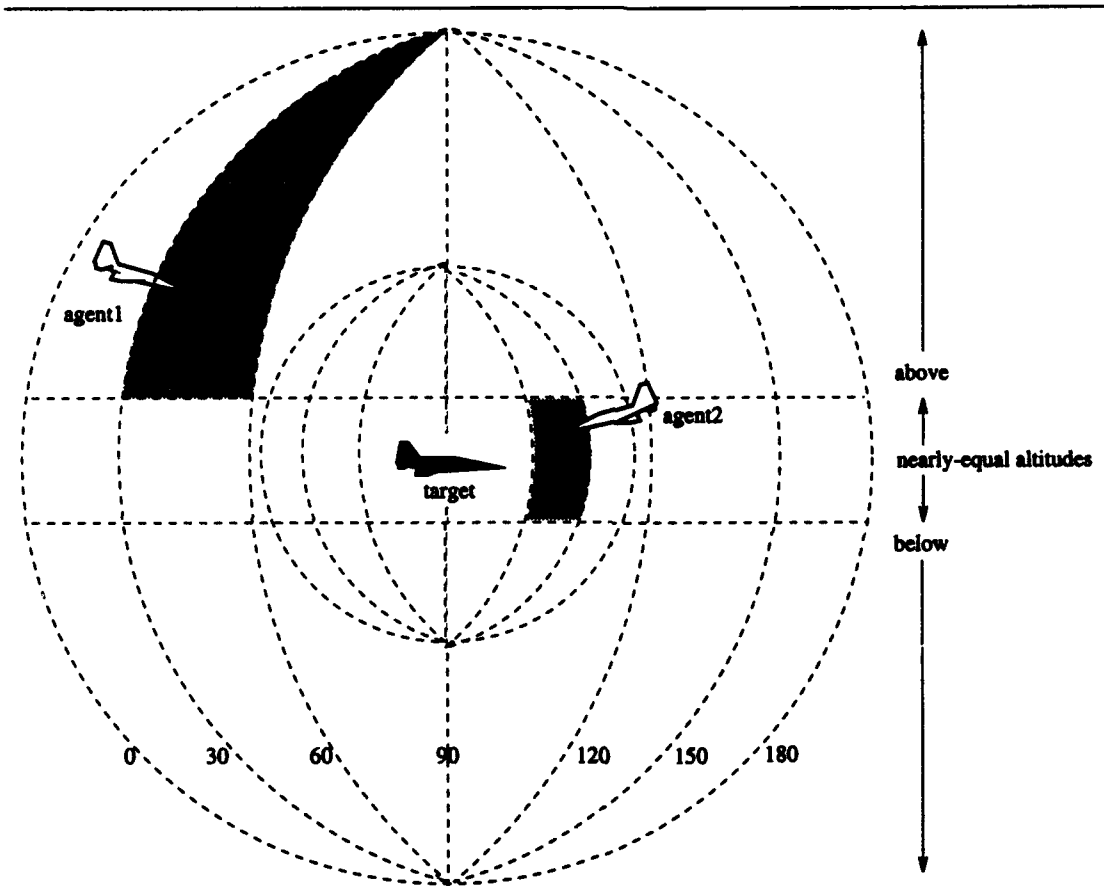

Figure 4.6 Plan sector combined view.

Code was added to MAXIM to compute the values of the array indices using these five parameters. The element located at the array position specified by the computed indices represented the best response uncovered during the discovery process for that particular plan sector. A response for any given sector is a list of the following format:

```
((nose-offset wing-offset tail-offset) percent-power)
```

The nose, wing, and tail offsets identified a point in space relative to the local target coordinate system with the origin centered on the current target aircraft position. The X axis of the local coordinate system was aligned with the vector resulting from the X and Y components of the target velocity vector. The local Z axis was parallel to the Z axis of the world coordinate system, and the direction of the local Y axis was computed by taking the cross product of the local X and Z axes. Once the local coordinate system had been established, the offsets in the selected response became the X, Y, and Z coordinates in the local system identifying the desired fly-to point. Finally, the local fly-to point was rotated back onto the world coordinate system to become the goal position for the flexible agent. The final response term, percent-power, was a real number in the range -1 to $+1$ which specified the percentage of the maximum deceleration or acceleration to be applied to the agent's velocity vector.

Using a local coordinate system rotated onto a portion of the target aircraft's velocity vector allowed the fly-to point to be positioned relative to the heading of the target aircraft. The rotation did not entirely account for target heading, however, since only two of the three target vector components were combined to form the nose-offset axis. The underlying reason for this approach was to preserve the intended effects of each of the offsets: the nose offset specifies how much to lead or lag the target, the wing offset specifies how far off the left or right target wing to fly, and the tail offset specifies how much above or below the target the agent is to fly. If the local offset coordinate system were allowed to rotate freely with the target aircraft velocity, wing, and tail axes then it would be possible for the offsets to have an opposite effect than was intended. For example, if the target drone were inverted, then a positive tail offset would result in a fly-to point beneath, instead of above, the target.

Although each 30 degree increment in aspect angle could have been used to select a response tuned for that sector, I instead decided to cut the number of sectors to be learned in half through mirroring. NOSTRUM was programmed to test candidate responses in the sectors contained within aspect angles 0 to 180 degrees; responses for aspect angles greater than 180 degrees were transformed by simply changing the sign of the wing offset, effectively flipping the fly-to point from one side of the target to the other. In total, there were 144 responses for NOSTRUM to learn during the discovery process, but 288 responses are available for a flexible agent to draw upon during an engagement.

4.3.4 Dead Zones. It became apparent during the initial testing of NOSTRUM that my scheme of adjustable fly-to points had some serious side effects. On several occasions, NOSTRUM would explore responses that placed the fly-to point behind the target by some amount when the agent advanced on the target from the rear. At some point during these simulations, the agent would actually reach and fly beyond the fly-to point. The moment the agent flew slightly beyond the fly-to point it would begin turning back towards the fly-to point, which was still behind the target, in spite of the fact that the target aircraft was directly in front of the agent. Other times, the agent would oscillate back and forth or up and down after it had passed the fly-to point as it tried to return to that position in the airspace.

This behavior forced me to change my perspective of the fly-to point. Instead of viewing it as a point in space that the agent wanted to reach and remain at, I decided to treat each new fly-to point as a *goal.* When the agent moves into a new plan sector, the fly-to point is assigned and becomes the goal of the agent as long as it stays within that sector. Once the agent flies within a certain distance of the fly-to point, which I called the *dead zone*, the system considers that goal to be achieved. At that point, and until the agent moves into a different plan sector when a more appropriate fly-to point can be selected, the agent flies totally dead-stick. During dead-stick flight, the agent moves at constant velocity in the heading it was traveling when it entered the dead zone.

4.4 NOSTRUM *Components*

The learning portion of the system developed during this research, NOSTRUM, was written entirely in the CLOS. Although NOSTRUM uses only a single object class for its processing, MAXIM uses objects and methods frequently. Staying within the constructs of a single language simplified the integration of the two systems.

Code integral to NOSTRUM can be cleanly divided into the following functional areas: instrumentation, heuristics, and the agenda. This section describes the major features of each functional area.

4.4.1 Instrumentation. A substantial amount of code was written to collect data during each simulation for later analysis by NOSTRUM. During each flexible agent update cycle, various features of the simulation are checked and flags may be set to indicate that a particular event has occurred. The instrumentation software tallies the amount of time that each aircraft could fire a missile upon the other, and NOSTRUM examines several other data points that can provide an indication that progress is being made developing a response. By the end of a scenario, NOSTRUM has the following data available to assess the worth of the response just tested:

- **within-front-hemisphere** If set to True, the agent was within the front hemisphere of the target aircraft sometime during the simulation.
- **within-rear-hemisphere** If set to True, the agent was within the rear hemisphere of the target aircraft sometime during the simulation.
- **shot-possible** If set to True, the agent was simultaneously within range and at the proper nose angle to take a shot on the target aircraft sometime during the simulation.
- **in-danger** If set to True, the target was simultaneously within range and at the proper nose angle to take a shot on the agent aircraft sometime during the simulation.
- **fly-by** If set to True, the agent was in the target rear hemisphere at some point during the simulation but flew past the target and into its front hemisphere.

- **end-game-shot-possible** If set to True, the agent was simultaneously within range and at the proper nose angle to take a shot on the target aircraft in the end-game position.
- **end-game-aspect-angle** The aspect angle of the agent relative to the target in the end-game position.
- **end-game-nose-angle** The angle between the agent's nose vector and line-of-site vector to the target in the end-game position.
- **end-game-velocity-delta** The difference in magnitudes of the agent and target velocity vectors in the end-game position.
- **end-game-range** The distance between agent and target in the end-game position.
- **time-while-shot-possible** The total amount of time that the agent was able to fire upon the target during the simulation.
- **estimated-shot-time** An estimated amount of remaining shot time if the simulation were allowed to continue.
- **time-in-danger** The total amount of time that the target was able to fire upon the agent during the simulation.

The definition of each instrumentation data point is fairly obvious, with the exception of *estimated-shot-time*. During testing, I realized that NOSTRUM was avoiding responses that placed the agent in an excellent end-game position because the actual response *time-while-shot-possible* was less than the shot time experienced using a different response. Had the simulation continued, however, these responses would have accumulated additional shot time because the agent was directly behind the target at nearly the same velocity. Estimating the shot time was a mechanism that forced NOSTRUM to explore responses in this category.

The shot time estimation function took into account four possible scenarios. The agent could have been in position for a final shot and moving faster than the target, in position for a final shot and moving slower than the target, too far away for a shot but moving faster than the target, or too close for a shot but moving moving slower than the

target. In each case the velocity was assumed to be constant, although in all likelihood it would probably change as a result of a non-zero power setting. It was also assumed that the agent would remain in position for a shot until it either got too close or too far away from the target. Using these assumptions, calculating estimated shot time was simply a matter of dividing the remaining distance before the agent was not within range for a missile shot by agent velocity. Since there are a number of unknowns that could adversely impact the estimation, NOSTRUM is conservative and applies only 50% of the estimated shot time. The total shot time associated with each response then is the sum of its *actual* and *estimated* shot times.

NOSTRUM uses the instrumentation data to compare the effectiveness of each response tested during the discovery process. To do this, NOSTRUM recalls the best features encountered during experimentation and compares them against values collected from the parent response and the response that is currently best. The determination of "best" is heuristically determined by examining the data: it is desirable to maximize some characteristics of a simulation while simultaneously minimizing others. For example, NOSTRUM tries to discover responses that increase the shot time of the flexible agent on the target, but it also tries to decrease the shot time of the target on the agent. NOSTRUM maintains a record of the following best features:

- **best-time-while-shot-possible** The greatest time experienced that the agent could fire upon the target while exploring possible responses for the current plan sector.
- **best-time-in-danger** The least time experienced that the target could fire upon the agent while exploring possible responses for the current plan sector.
- **best-end-game-aspect-angle** The smallest aspect angle experienced while exploring possible responses for the current plan sector.
- **best-end-game-nose-angle** The smallest nose angle experienced while exploring possible responses for the current plan sector.

- **best-end-game-velocity-delta** The smallest absolute difference between agent and target velocities experienced while exploring possible responses for the current plan sector.

4.4.2 Heuristics. Within NOSTRUM there were heuristics to assess the quality of a response, spawn new responses, and determine when exploration for a particular response should stop. The heuristics were grouped by function into *interestingness*, *response*, and *termination* heuristics. Although the heuristics contributed only a tiny fraction to the program code size, minor changes in any one of the heuristic sets often had a profound impact on the quality of the response discovered by NOSTRUM.

4.4.2.1 Interestingness Heuristics. The purpose of NOSTRUM's interestingness heuristics is to examine the results of a simulation using a particular response and assign a numeric measure of worth to the response. Interestingness is based not only on the actual results produced by a response but also on some expectation of success if exploration is continued from that response. The determination of what is good using these heuristics is not guaranteed to be always accurate, but this is precisely why they are called "rules of thumb." Most of the time the heuristics will work and can make the discovery process more efficient by leading the system towards responses that appear more promising and away from search paths that tend to be fruitless.

Constructing NOSTRUM's interestingness heuristics was a difficult task. The purpose of the system was to find responses that improved the offensive position of the agent while at the same time improving its defensive position. Some attributes, such as increasing shot time or decreasing time in danger, clearly stood out as an indication that one response was better than another. The greater problem was identifying responses that had potential but were not very good in and of themselves. In NOSTRUM, the best way for the agent to maximize shot time against the target aircraft is to get in position behind the target, in range and at the proper track angle for a missile shot, with the agent flying at nearly the same velocity as the target aircraft. This analysis generated three additional parameters for NOSTRUM to use when it assessed the potential of a response: end-game aspect angle, nose angle, and the delta between agent and target velocities.

The end-game aspect angle establishes the final position of the agent with respect to target heading. A smaller aspect angle indicates that the agent is more to the rear of the target, but it gives no indication of agent heading relative to the target. This information is provided by the end-game nose angle, which measures the angle between the agent's velocity vector and the line-of-sight vector from agent to target. A small aspect angle combined with a small nose angle is a desirable end-game position, because it means that the agent was behind and heading towards the target. An end-game position is further improved by reducing the delta separating agent and target velocities. When the two velocities are the same and neither aircraft is accelerating (or decelerating), a good end-game position can be maintained indefinitely.

In NOSTRUM, interestingness is a positive quantity calculated by summing up bonus points assigned for certain quantitative and qualitative characteristics observed while testing a response. The bonus points are then added to the interestingness of the parent response so that the measurement always increases or remains constant from parent to child. The format of each NOSTRUM interestingness heuristic is an if-then type statement of one of the two following forms:

```
IF <some event occurred> THEN <add a bonus>
IF <an improvement occurred> THEN <add a bonus>
```

NOSTRUM uses eleven heuristics to calculate the increase in a response's interestingness over that of its parent. The complete set is shown below, in pseudo-code format:

- **IH1** If the agent was able to take a shot sometime during the simulation add 5 bonus points.
- **IH2** If the agent was able to take a shot in the end-game position add 10 bonus points.
- **IH3** If the agent was never in danger during the simulation add 2 bonus points.
- **IH4** If the time the agent is able to take a shot is greater than the parent response shot time add 25 bonus points.

- **IH5** If the time the agent is in danger is less than the parent response danger time add 10 bonus points.
- **IH6** If the end-game aspect angle is less than the parent response aspect angle and the agent did not fly by the target add 2 bonus points.
- **IH7** If the end-game nose angle is less than the parent response nose angle and the agent did not fly by the target add 2 bonus points.
- **IH8** If the end-game velocity delta is less than the parent response velocity delta add 2 bonus points.
- **IH9** If the time the agent is able to take a shot is greater than the parent response shot time, and the time the agent is in danger is less than the parent response danger time, add 15 bonus points.
- **IH10** If the end-game position improved over the parent in aspect angle, nose angle, and velocity delta add 5 bonus points.
- **IH11** If the response is the best response so far add 10 bonus points.

Bonus values were assigned based on my impressions of factors that might indicate increasing response quality. For example, the greatest bonuses went to responses that improved total shot time (25 points), or both total shot time and danger time (15 points), over that of the parent response. Reducing the time in danger was also worthy of a bonus (10 points), but it was not as interesting as an improvement in shot time. The disparity in bonuses was a deliberate attempt on my part to force flexible agents to give a higher priority to the mission than to their own survival. A number of the interestingness heuristics looked for gradual improvements in the response (decreasing angles, decreasing velocity deltas) that might not provide an immediate increase in shot time, but had the potential for producing a better child response. These improvements were de-emphasized, as indicated by the marginal bonuses associated with them, but the bonuses were enough to force NOSTRUM to continue exploring response evolution along that path. If a bonus was not awarded for these minor improvements, NOSTRUM would have inserted the child responses into the agenda behind all responses with the same interestingness. These branches in the

search tree might never get explored because the system discontinued exploration before reaching them.

4.4.2.2 Response Heuristics. A set of response heuristics, operating independently of the interestingness heuristics, was used to suggest modifications to a response. The intent is to modify a response in such a way that it improves the behavior of the agent. Improvements are gauged by the impact a modification has had on the *time-while-shot-possible* and *time-in-danger*, as well as the *end-game-aspect-angle*, *end-game-nose-angle*, and *end-game-velocity-delta.*

Care had to be taken while crafting the response heuristics. A criticism of some DBL systems has been that the system was well coached, lead by the heuristics to "discover" exactly what the programmer intended. Avoiding this problem with NOSTRUM was not entirely difficult, however, since my understanding of the domain was limited. In addition, the behavior resulting from interaction in the various responses as the agent passed through multiple plan sectors was not known during program development, further reducing the effect of personal bias in the heuristics.

NOSTRUM's response heuristics are similar in form to the interestingness heuristics. Each heuristic is an IF-THEN type construct checking for the existence of a certain condition, but instead of returning a bonus a response heuristic suggests a modification to the current response. The form of a response heuristic is shown below:

```
IF <condition> THEN <response modification>
```

Response modifications can change the location of the fly-to point by adding or subtracting an amount from the nose, wing, or tail offsets, or they can increase or decrease the power setting applied in that plan sector. Finally, through a process known as *spawning*, response heuristics that have their preconditions satisfied are recorded in a list and placed, along with the parent response, onto the agenda. Later, during response heuristic application, NOSTRUM will select a previously recorded response heuristic from the agenda and use it to generate a new child response.

NOSTRUM used nine heuristics to spawn new responses. The complete set is shown below, in pseudo-code format:

- **RH1** If the agent didn't fly past the target and a shot was possible in the end-game position, then try moving the fly-to point beneath the target aircraft. *Rationale: Trade altitude for airspeed and try to get as close to the target as possible, but within range for a missile shot.*
- **RH2** If the agent didn't fly past the target and a shot was possible in the end-game position, then try moving the fly-to point ahead of the target aircraft. *Rationale: Use lead pursuit to reduce the agent to target separation, but stay within range for a missile shot.*
- **RH3** If the agent didn't fly past the target and a shot was possible in the end-game position, then try increasing the power setting. *Rational: Increasing acceleration will increase velocity and reduce agent to target separation, but don't get so close that a missile shot is prohibited in the end game position.*
- **RH4** If the agent did fly past the target or a shot wasn't possible in the end-game position, then try moving the fly-to point above the target aircraft. *Rationale: Trade airspeed for altitude and try to increase agent to target separation.*
- **RH5** If the agent did fly past the target or a shot wasn't possible in the end-game position, then try moving the fly-to point behind the target aircraft. *Rationale: Use lag pursuit to increase agent to target separation.*
- **RH6** If the agent did fly past the target or a shot wasn't possible in the end-game position, then try decreasing the power setting. *Rationale: Increasing deceleration will decrease velocity and increase agent to target separation.*
- **RH7** If the agent was in the front hemisphere of the target aircraft, then try moving the fly-to point off the right wing of the target. *Rationale: Instead of flying at the target try to fly a more parallel course. This might place the agent in a better position to swing back around and close in on the target from the rear.*

- **RH8** If the agent was in the front hemisphere of the target aircraft, then try moving the fly-to point behind the target. *Rationale: Get out of the line-of-sight of the target as soon as possible.*
- **RH9** If the agent was in the front hemisphere of the target aircraft, then try moving the fly-to point off the left wing of the target. *Rational: Instead of flying at the target try to fly a more parallel course. This might place the agent in a better position to swing back around and close in on the target from the rear.*

4.4.2.3 Termination Heuristics. A third set of heuristics was used by NOSTRUM to determine when to halt a simulation in progress. Each simulation was run to test the quality of a potential plan sector response, but for obvious practical reasons a simulation could not go on indefinitely. A mechanism was needed to stop the simulation at some point during the test.

One way to cap simulation run time is to set a maximum time and stop the simulation when it runs beyond this limit. Care must be taken when setting the maximum time to ensure that it will allow a good measure of response performance. If the time span is too short, responses that are potentially good may go unnoticed because the agent did not have enough time to move into its end-game position. On the other hand, a poorer quality response can degrade the agent's position well before the maximum simulation time has elapsed.

Detecting degrading agent position is accomplished in NOSTRUM with the *fly-by* flag. When the fly-by flag is set, it indicates that the agent was previously in the rear hemisphere of the target but flew beyond and into the target's front hemisphere. At this point the agent's position can only get worse, and the agent will probably not have enough time left in the simulation to come back around for a second shot on the target. Once a fly-by occurs it is also likely that the simulation will no longer be judging the quality of the response the system was supposed to test, but will instead be exercising a number of other responses.

NOSTRUM uses both a maximum cap on run time and fly-by detection to determine when a simulation should stop. The maximum run time prevents the system from testing

a response indefinitely, and fly-by detection stops the simulation when it appears that the agent's position will worsen instead of improve.

4.4.3 The Agenda. The agenda is a critical component of a DBL system. In NOSTRUM, the agenda is used to maintain a list of responses to try next, rank-ordered by interestingness. Responses which seem to have the greatest potential will be at the head of the agenda, while responses that are expected to produce poorer quality results move towards the end of the agenda. Each response tested by NOSTRUM is represented as a *response object.*

4.4.3.1 Response Objects. The response-object is the only object class used within NOSTRUM. A response-object is created for each response evaluated by NOSTRUM, and it encapsulates the important details of a potential response using the following slots:

- **response** A list representing the response tested by NOSTRUM for the plan sector.
- **interestingness** After a response has been tested, its interestingness is calculated and stored in this slot.
- **time-while-shot-possible**
- **estimated-shot-time**
- **time-in-danger**
- **fly-by**
- **end-game-aspect-angle**
- **end-game-nose-angle**
- **end-game-velocity-delta**
- **end-game-shot-possible**
- **sectors-used** This slot contains a list of all the sectors that the agent passed through during the course of the entire simulation.
- **evolution** This slot lists the evolutionary path of the response, identifying the sequence of response heuristics that produced it (This was not needed by NOSTRUM,

but was used during program development to track the evolutionary history of a response).

- **ancestors** This slot lists all of the response objects that are ancestors of the response.

When a response-object is created, the *response* slot is set to the response to be tested, and the *evolution* and *ancestors* slots are updated to reflect the origins of the new response. The remaining slot values are undefined until after the response has been selected from the agenda and tested in an actual simulation. Following testing, instrumentation data collected during the simulation is used to fill in the appropriate slots.

4.4.3.2 Agenda Structure. Since only a single agenda is used by NOSTRUM, I chose not to implement the agenda as a CLOS object. There did not appear to be much utility in creating a separate class and methods for an agenda object when there was no possibility for multiple instantiations. Instead, the agenda is a simple list structure. Items with the greatest interestingness are at the head of the list, while the lowest interestingness items are sorted to the tail of the list.

Each element in the agenda is a list of the following form:

```
((interestingness parent-object (RHa RHb ... RHz)))
```

The *interestingness* is used to place items in the agenda list in order of increasing interestingness. This value is actually the interestingness calculated for the response contained in the *parent object.* Each parent is a *response-object* that has been selected from the agenda and tested in a simulation. Following the simulation, it is possible that the performance of the parent response satisfied the preconditions of one or more response heuristics, which will result in a set of child responses. Child responses are not actually generated until the moment before they are tested in a simulation, so a list of all response heuristics found to be applicable is maintained in a list associated with the parent and parent's interestingness.

4.4.3.3 Preventing Repeat Experiments. A characteristic of NOSTRUM's discovery process is that multiple search paths can generate the same response. Since each response heuristic modifies the parent response by adding or subtracting an offset, it makes no difference which order a set of response heuristics has been applied; offset application is totally commutative. For example, the heuristic evolution (RH1 RH1 RH5) will produce exactly the same response as the heuristic evolution (RH1 RH5 RH1).

To prevent the system from re-exploring previously tested responses, NOSTRUM maintains a record of *responses-already-tried.* When NOSTRUM selects a new response to explore from the agenda, it first verifies that the response doesn't already appear in the *responses-already-tried* list. If it does, the response is skipped and another selection is made.

4.4.4 Response Selection, Evaluation, and Generation. Figure 4.7 illustrates the process of selecting a response from the agenda, evaluating the performance of that response, and generating new responses using a hypothetical set of agenda items. This figure will be referred to frequently in the following discussion.

The first step in the cycle was to select the next task from the agenda, indicated in the figure as Step 1. NOSTRUM examined the list of remaining response heuristics associated with the most interesting parent which, in this case, was the list (RH1 RH3 RH5). When parent1 was previously explored by the system, the preconditions of response heuristics RH1, RH2 and RH3 were satisfied, and the three heuristics were recorded then for later application. At Step 2, the response heuristics were dynamically arranged so that the heuristic that was working best was applied first. In the figure, RH3 was producing the best results, so it was selected at Step 3.

At Step 4, RH3 was applied to the parent1 to produce the next response. Application of a response heuristic entailed adding a positive or negative delta to one of the four values in the response, ((nose-offset wing-offset tail-offset) percent-power). In the case of RH3, the response was modified by adding a positive delta to the power setting. A child response object, child1, was created, and the modified response was inserted into the corresponding object slot, and into the universal plan response table. NOSTRUM

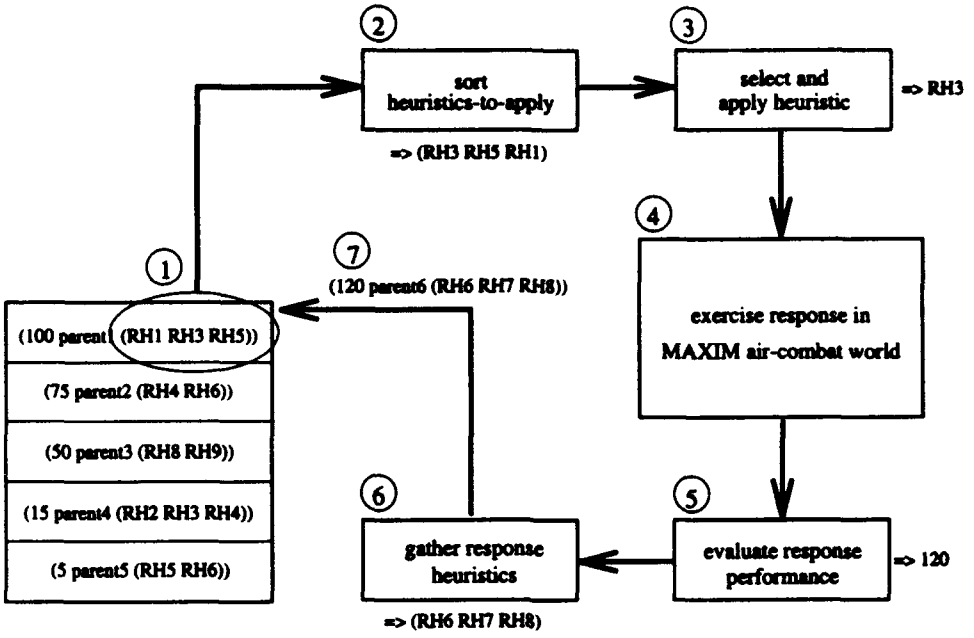

Figure 4.7 NOSTRUM **operational sequence.** An illustration of the cycle repeated while NOSTRUM explores responses for a plan sector using a hypothetical set of agenda items.

dynamically created a scenario to exercise the response, and control was passed to MAXIM at Step 5 to measure performance using the scenario.

At Step 6, the MAXIM simulation ended and NOSTRUM resumed control to evaluate performance. Evaluation was accomplished by calling each of the interestingness heuristics in turn and summing up the values returned by each. The final sum was added to the interestingness of the parent to become the total measure of response interestingness. Evaluation measurements were stored in the slots of the `child1` response object, and NOSTRUM moved on to the task of gathering response heuristics at Step 7.

Child responses are not actually created until Step 4 in a subsequent cycle, one step prior to response testing. However, NOSTRUM still had to determine which response heuristics might be able to improve a response and record them for later application. The preconditions of each response heuristic were tested, and those that were satisfied

were stored by name in a list. At Step 7, **child1** itself became a parent object, **parent6**. NOSTRUM created a **parent6** agenda item from the response and interestingness of **child1**, and the list of applicable response heuristics. Finally, at Step 8, the agenda item was inserted into the agenda at the proper location based on its interestingness.

When a child response had the same interestingness as the parent it was an indication that the child did not improve the overall performance of the agent. Rather than investigate this child immediately, it was inserted into the agenda behind the parent. Adding items to the agenda in this manner ensured that each response heuristic applicable to the parent had a chance to be selected as the next task before moving on to the tasks beneath it. Only when the interestingness of the child was greater than the parent did exploration of the parent's offspring temporarily halt.

4.5 NOSTRUM *Operation*

NOSTRUM operates in a continuing cycle of selecting, testing, evaluating, and generating responses. This cycle repeats until NOSTRUM determines that progress is no longer being made, at which point exploration stops.

The learning loop is set up to explore responses for all 144 plan sectors individually. NOSTRUM first tries to improve upon all the plan sectors in the close-in range before moving out to the long-distance range, since scenarios with the agent positioned initially at the long-distance range will eventually activate the close-in plan sectors. Within each range division NOSTRUM then explores responses starting from the rear of the target aircraft and working towards the front. This is done for the same reason as starting from close-in ranges and working outward: since the goal of the agent is to get in position behind the target and maintain missile lock, plan sectors to the rear of the target will probably be activated when learning the frontal plan sectors. Within each plan sector, NOSTRUM refines responses further by first modifying agent velocity, followed by heading-crossing angle, and finally agent altitude relative to the target. Working from the outer learning loop in, responses are then explored in the following order:

`ranges → aspects → altitudes → heading-crossing-angles → velocities`

When NOSTRUM begins learning, the agenda is completely empty and the table of attack responses is filled with the default for every plan sector, the default response being `((0 0 0) 0)`. Counters for each of the five attack response table indices are reset to zero, and learning begins for the first plan sector. The default response is the first response tested in every plan sector because the agenda will always be empty when the system starts learning a new response. Once the first experiment is run, however, additional responses will be spawned and added to the agenda for selection in future experiments. NOSTRUM experiments with various response for the plan sector, and eventually returns the best response it could find. The response is "learned" by inserting it into the attack response table at the proper location, and then the entire table is written to a disk file. The inner-most index counter is incremented and the process repeats for the next plan sector, until all 144 responses have been learned.

4.5.1 Plan Sector Divisions. As mentioned in Section 4.3.3, the response selected by a flexible agent during the attack phase is based on five key parameters identified during domain analysis. During implementation, it became necessary to specify the range of values that would correspond to a value suitable for indexing into the array of responses. Table 4.1 shows the various values for each of the parameters and the corresponding index into the table of attack responses.

Although the process of selecting values for the plan sector divisions was not a rigorous one, values were selected with the desired agent behavior in mind and my perception of the divisions that might make this behavior possible.

4.5.2 Dynamic Scenarios. A necessary element of the NOSTRUM learning approach is dynamic scenario creation. Learning a response for each plan sector requires the system to set up a situation that guarantees that the sector under scrutiny will be activated. The easiest way to do this is to initialize the simulation with the agent in a position relative to the target that places it in the desired plan sector. The target aircraft always has the same initial position, bearing, and velocity, but the position, heading and

Plan Sector Divisions			
Parameter	Array Index	Minimum	Maximum
Range	0	0	6000
	1	6000	BIGGEST-NUM
Aspect angle	0	0	30
	1	30	60
	2	60	90
	3	90	120
	4	120	150
	5	150	180
Altitude delta	0	-2000	2000
	1	2000	BIGGEST-NUM
	2	SMALLEST-NUM	-2000
Heading crossing angle	0	0	90
	1	90	180
Velocity delta	0	0	BIGGEST-NUM
	1	SMALLEST-NUM	0

Table 4.1 **Plan sector divisions.** This table illustrates how NOSTRUM computes the index into the attack response table based on the five key parameters.

velocity of the agent relative to the target are adjusted to match the plan sector. For example, if the system is learning a response for close-in range, aspect angle between 30 and 60 degrees, altitudes relatively equal, a heading crossing angle indicating agent and target heading in nearly the same direction, and an agent velocity greater than the target, then NOSTRUM creates the following scenario dynamically (This training scenario is depicted graphically in Figure 4.8):

Range to target:	5000 m
Aspect angle:	45 degrees
Altitude difference:	0 m
Heading crossing angle:	0 degrees
Velocity difference:	150 m/s

Table 4.2 **Sample training scenario.** A training scenario dynamically created for learning the response accessed by range index 0, aspect index 0, altitude index 0, heading crossing angle index 0, and velocity index 0.

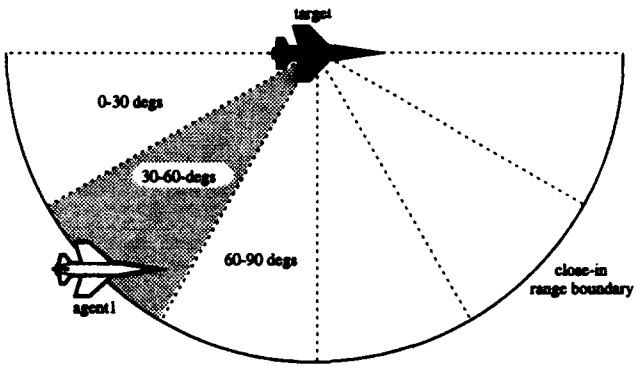

Figure 4.8 **Graphical depiction of a training scenario.**

To prepare the dynamic scenarios, NOSTRUM uses several lists containing training scenario settings. These settings define the initial values of each of the five key parameters used to index the table of attack response. The program then uses these values to compute actual (X Y Z) coordinates for the agent start position and vectors that occupy the MAXIM F15 object slots. Table 4.3 lists the values used to generate each of the training scenarios.

Because responses were tuned for specific situations, I expected there to be a certain amount of brittleness in the responses learned by NOSTRUM. This issue is addressed in Chapter 5.

4.5.3 Heuristic Ordering. Responses are spawned in the order in which the heuristics are called, and this can have a marked impact on both search efficiency and quality of solution produced by the system. For example, if a particular heuristic continually produces child responses of higher quality than the parent response, then it would be desirable to have responses resulting from this heuristic tested first. Similarly, those heuristics that have been producing lower quality children should only be used as a last resort since they have a history of worsening the situation. Without a mechanism to allow this, heuristics would instead be applied in the order in which they were coded. Response

Training Scenario Values		
Parameter	Array Index	Training Value
Range	0	5000
	1	15000
Aspect angle	0	15
	1	45
	2	75
	3	105
	4	135
	5	165
Altitude delta	0	0
	1	$0.5 * Range$
	2	$-0.5 * Range$
Heading crossing angle	0	0
	1	180
Velocity delta	0	150
	1	-150

Table 4.3 **Training scenario values.** This table lists the initial settings for the training scenarios based on variation in each of the five key parameters.

heuristics that generated terrible results would always be explored first if they appeared at the head of the list, wasting time and possibly leading the system down a dead-end path.

The DBL system MAVERICK experienced the same effect of heuristic ordering. The solution to minimize the effects of heuristic ordering was to dynamically sort heuristics to give precedence to those that had been improving solution quality recently. The same scheme was implemented in NOSTRUM to sort the set of response heuristics prior to their application. Each time a heuristic produced a response with a higher interestingness than the parent response, that heuristic received a bonus equal to the difference in interestingness from child to parent. The scores associated with each response heuristic were used as keys to a sort routine, which arranged the heuristics in order of decreasing scores.

4.5.4 Hill Climbing. A characteristic of many AI systems is the use of a hill climbing algorithm to search a solution space. The phrase "hill climbing" captures the tendency of a system to constrain its search around a local maximum when a better solution might be found elsewhere. Using the hill climbing strategy, a system generates

and tests possible solutions; when the system finds a generated solution that is better than the current one, then this solution becomes the current solution, and the cycle repeats. Eventually, the system will either find a solution that is the goal state, or it will reach a peak in solution quality that cannot be improved upon without first exploring poorer solutions. In both circumstances, the final solution is not guaranteed to be optimal unless each point in the search space is on a positive quality gradient leading the system towards a single maximum. Figure 4.9 illustrates an ideal hill climbing scenario where the global maximum could easily be found.

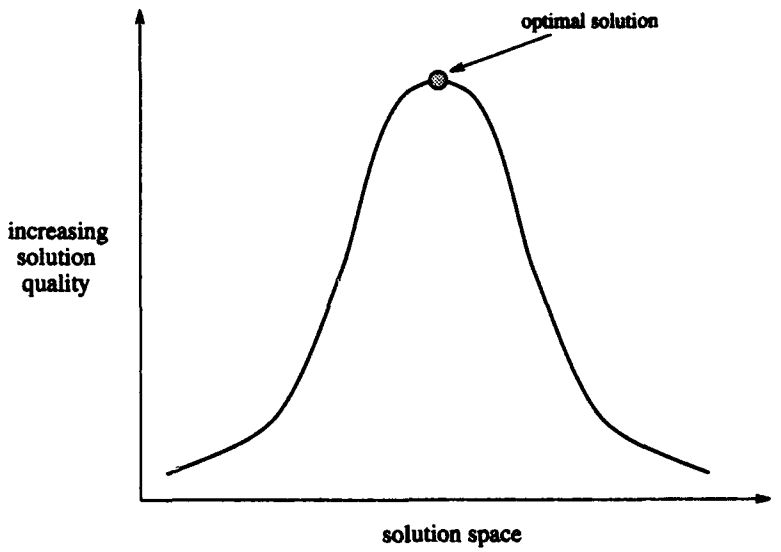

Figure 4.9 **Optimal hill climbing.** A scenario where hill climbing is guaranteed to find the global maximum.

NOSTRUM's discovery process can also be characterized as an exercise in hill-climbing, although the optimal solutions are much more difficult to find. Figure 4.10 presents a hypothetical scenario marked by several local maxima and a plateau, a more likely situation encountered by NOSTRUM as it explores possible maneuver responses. Should NOSTRUM begin exploration at the left side of the graph, it would first encounter the local maximum at Point 1. To get to the best solution at Point 3, NOSTRUM would first have to explore

poorer quality solutions to the right of Point 1. If exploration continued, solution quality would level out for a time near Point 2 before finally reaching its maximum value at Point 3.

Figure 4.10 actually over-simplifies the events that occur while NOSTRUM is exploring potential responses. NOSTRUM measures improvements in solution quality as an increase in *shot-time* or a decrease in *time-in-danger*, but additional measurements are used to help determine when progress is being made. As mentioned in Section 4.4.2.1, the interestingness heuristics take many other features into account when the interestingness of a response is computed. Responses are selected and tested on the basis of their interestingness, so exploration rarely proceeds along a series of adjacent solutions in the search space. Instead, NOSTRUM is more likely to hop from point to point, investigating lower quality solutions when nothing better is available and abandoning them (at least temporarily) when a better solution comes along.

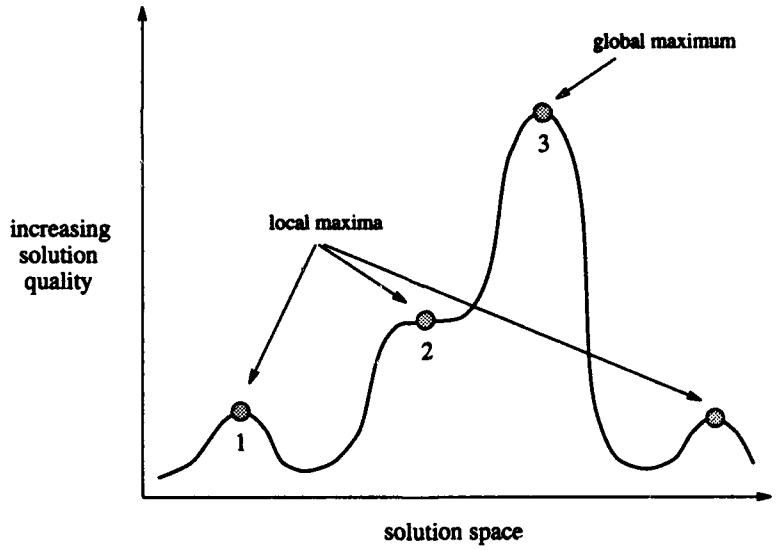

Figure 4.10 **Realistic hill climbing.** A more probable scenario encountered by NOSTRUM, marked by several local maxima and a plateau.

4.5.5 Plateauing. In the initial stages of development, NOSTRUM used a simple counter to halt exploration when a preset number of responses had been tested. Another strategy I used was to stop exploration after a preset amount of time had elapsed. The

problem with both approaches, however, is that they do not take into account the quality of responses being explored. The system may uncover the best response it could possibly find in the first cycle, but would instead be forced to continue exploration for some additional number of cycles or period of time. Likewise, NOSTRUM might be in the process of significantly improving a response but could be cut off when either of these arbitrary boundaries was crossed, a phenomena known as the *horizon effect.* Using a learning cycle counter or timer does not allow the system to decide for itself when a point of diminishing returns has been reached.

Instead, NOSTRUM uses a *plateau counter.* The plateau counter is reset to zero every time a response is found that has the best shot (ST) and danger times (DT). This counter is incremented for each response tested that does not improve shot or danger times. After the plateau counter reaches some preset value (set in the current version of NOSTRUM to 20), exploration for that response terminates and the system goes on to learn the next response.

It is understood that using a plateau counter to halt learning has its disadvantages. There is the possibility that NOSTRUM would discover an improved response at plateau count 21, but the system was prevented from finding it because it was stopped at plateau count 20. This situation could arise no matter what value n was assigned to the maximum number of plateaus because there would always be the possibility of a better solution existing at count $(n + 1)$. However, as with all heuristics, it is expected to work *most* of the time.

4.5.6 Side-Effects of Learning. As NOSTRUM explored possible responses for each plan sector, it was necessary to run a scenario to evaluate the performance of each response. As each scenario executed, there were several undesirable side-effects that could occur, impacting the quality of the learned response.

When the system was initialized just prior to learning, NOSTRUM set each plan sector response to its default value, `((0 0 0) 0.0)`. Since the fly-to point offset in the default response was zero, the flexible agent was programmed to behave like a standard MAXIM agent using proportional control. NOSTRUM then began the process of improving the

responses for each sector, one by one, until alternative responses for all 144 plan sectors had been explored. As NOSTRUM explored responses for an individual plan sector it sometimes flew into and activated the default response in sectors where learning had not yet taken place. NOSTRUM eventually learned more appropriate responses for these plan sectors, but the default response may have already had an impact on the results in other plan sectors.

There was no way to eliminate the incidental activation of default responses, but its impact could have been minimized by repeating the learning process for a number of cycles. The greatest impact of incidental activations should occur during the first learning cycle because default responses will be used instead of another, more appropriate response. Subsequent learning cycles may continue to activate plan sectors that have not yet been refined, but the impact of each should decrease as the the responses begin to settle to their steady-state values. At some point, additional learning cycles should produce such insignificant changes that the system can be considered in equilibrium. Although this approach was considered it was not investigated.

Learning responses for plan sectors that were not to the immediate rear of the target resulted in another unpleasant consequence: the difficulty of assigning credit where credit is due. When NOSTRUM first began exploring candidate responses for plan sectors at the rear of the target aircraft, most of the simulation time was spent in one or two plan sectors. As the aspect angle increased, however, more and more plan sector boundaries were crossed as the agent maneuvered around the target, making it increasingly difficult to determine which plan sector response contributed most to the overall agent performance. NOSTRUM was not equipped to deal with the credit assignment problem effectively because it explored responses for each plan sector individually. After learning a plan sector response, NOSTRUM did not review it later on to suggest changes that could improve the performance of other responses. Instead, NOSTRUM was coded using the assumption that credit for success or failure of a response being tested belongs entirely to that response, and that previously learned responses activated along the way provide the correct behavior. This is a heuristic used in everyday life. When we learned to drive a car, we probably first practiced starting the engine in the driveway and tested the effect of pressing on the accelerator. Later, a parent took us out for a drive around the neighborhood, and soon we were driving on busy

city streets. Each period of learning built on previously learned knowledge. Of course, there are times when previously learned knowledge proves to be incomplete, such as knowing to steer into a turn while skidding, or resisting the temptation to lock the brakes on an icy street. A universal plan to drive a car could perform well most of the time, even if we failed to train for these specialized conditions.

Assigning success and failure completely to the current response was made more palatable by a slight shift in perspective. Responses learned by NOSTRUM for plan sectors to the rear of the target aircraft will be the most immune to credit assignment difficulties because they will seldom activate other responses. These responses are likely to be the highest quality responses discovered by NOSTRUM. As NOSTRUM moves out and begins learning other responses, incidental activations will be more common and will have a greater impact on the responses finally learned. Rather than focusing in on them individually, discovery in these plan sectors can then be viewed as exploration for a response that puts the agent in a position to utilize the higher quality responses found towards the rear of the target. Plan sectors with the best responses will be referred to in later sections as *pure* plan sectors.

4.6 Summary

This chapter presented a detailed description of the NOSTRUM architecture, a program for improving reactive aircraft agent combat tactics through discovery-based learning. A necessary component of any DBL system is a world to explore. NOSTRUM's world was provided by MAXIM, an air combat simulator developed at AFIT for reactive autonomous air agents. Behavior of a MAXIM agent is generated by a static universal plan that is predictable and sometimes inappropriate. The first step towards enlarging the universal plan was to devise a suitable representation for additional responses, providing the motivation for a sphere of plan sectors surrounding the target aircraft. NOSTRUM tested variations in the agent's response for each plan sector and remembered those that work best.

NOSTRUM's discovery process was managed by an agenda and three sets of heuristics. The agenda maintained a list of responses sorted by their interestingness, a value computed to indicate the worth or a potential increase in worth of a plan sector response using

the first set of heuristics. Associated with each response in the agenda was a list of applicable response heuristics that were used to modify a response and guide the system towards a better solution. Discovery was characterized by a repeating cycle of selecting and generating a child response from the most interesting parent on the agenda, running a scenario to collect data, evaluating the response using interestingness heuristics, collecting response heuristics for later application, and inserting the new parent back into the agenda. When NOSTRUM sensed that additional exploration was no longer improving a response, it remembered the best response tested and moved on to the next plan sector.

V. Results & Issues

5.1 Introduction

This chapter presents an analysis of data collected during testing and evaluation of the NOSTRUM learning system. During testing, I collected data to evaluate the two sub-hypotheses presented as research objectives in Chapter 1. The results for each sub-hypothesis will be presented individually, starting with the DBL Hypothesis.

5.2 Testing the DBL Hypothesis

The DBL Hypothesis in Section 1.3 stated my belief that discovery-based learning could be used to incrementally improve the behavior of an autonomous agent. A system was proposed that could test and refine individual responses in each of the 144 plan sectors using discovery and remember those that appear to work best. This section presents and evaluates data collected while testing the DBL Hypothesis.

One disadvantage of the plan sector approach is that it is impractical to present the results from all 144 plan sectors. Instead, the analysis begins with a presentation of data collected while learning within a few interesting plan sectors, starting with the one that places the agent almost directly behind the aircraft. In Section 4.5.6 this was referred to as a *pure* plan sector.

5.2.1 Learning in Pure Plan Sectors. A pure plan sector identifies a starting position of the agent relative to the target where the agent is behind and to the rear of the target. These sectors are pure in the sense that an agent starting in this position will rarely cross a plan sector boundary as it tries to lock in position behind the target, if a boundary is crossed at all. Figure 5.1 depicts graphically the first training scenario to be discussed during this analysis. The scenario placed the agent almost directly behind, at close range, parallel heading, and at the same altitude as the target aircraft.

NOSTRUM internally identifies each plan sector with a string constructed from the five key feature indices into the attack response table. The plan sector shown being activated in Figure 5.1 is referred to by NOSTRUM with the identifier "0-0-0-0-0". Referring to Table 4.3

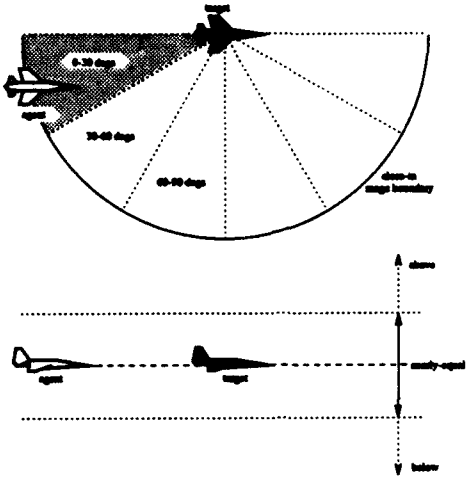

Figure 5.1 **Graphical depiction of training scenario for pure plan sector 0-0-0-0-0.**

for dynamic scenario specifications, NOSTRUM generated the dynamic scenario shown in Table 5.1 for exploring responses in plan sector 0-0-0-0-0.

Training Scenario for Sector 0-0-0-0-0		
Identifier Digit Position	Value	Meaning
1	0	Agent velocity is 150 m/s greater than target
2	0	Agent heading is parallel to target
3	0	Agent range to target is 5000 m
4	0	Agent altitude is equal to target's
5	0	Agent aspect angle to target is 15 degrees

Table 5.1 **Training scenario created for plan sector 0-0-0-0-0.**

The first action taken by NOSTRUM during response exploration for plan sector 0-0-0-0-0 was to run a simulation using the default proportional control response, which was set to `((0 0 0) 0.0)` in each plan sector. The results of that simulation became the baseline measurement of performance against which other plan sector responses could

be compared. When the simulation using the default response was completed, NOSTRUM generated a line of textual output displaying the important data points collected during the simulation:

			END-GAME POSITION						TIMES			
LC	PC	((NNNNN WWWWW TTTTT) AAAA)	ASP	NOS	VDEL	RANGE	FSH	FBY	ST	EST	DT	INT
1	0	((0 0 0) 0.0)	174	174	150	128	NIL	T	9	0	0	0

Table 5.2 **Sample** NOSTRUM **output.** Results of simulating the default response.

The textual line of output requires some explanation, beginning with the mnemonics identifying each column of data.

1. **LC:** The current learning cycle. This value increments by one for each response tested during discovery.
2. **PC:** The plateau count, representing the number of cycles that the best response discovered has remained the best response. This value normally increments by one for each response tested during the learning cycle, but will be reset to *0* when a new best response has been found.
3. **((NNNNN WWWWW TTTTT) AAAA):** The response being tested. The nose offset value was indicated by NNNNN, wing offset by WWWWW, tail offset by TTTTT, and power setting by AAAAA.
4. **ASP:** The aspect angle measured between agent line-of-site and target tail in the end-game position.
5. **NOS:** The nose angle measured between agent line-of-site and agent velocity-vector in the end-game position.
6. **VDEL:** Difference in agent and target velocity magnitudes. Positive quantities indicated that the agent was moving more quickly.
7. **RANGE:** The distance separating agent and target in the end-game position.
8. **FHS:** A boolean flag indicating whether or not a final shot was possible in the end-game position.

9. **FBY:** A boolean flag indicating whether or not the agent flew by the target aircraft.

10. **ST:** Total shot time accumulated during the simulation, including both actual and estimated shot times.

11. **EST:** Estimated shot time that the agent might have accumulated as actual time if the simulation had continued running.

12. **DT:** The time the agent was in danger of being fired upon by a missile launched from the target aircraft.

13. **INT:** The interestingness of this response.

5.2.1.1 Plan Sector 0-0-0-0-0 Results. Table 5.2 indicated that during the first learning cycle the default response was the best response explored so far. Since no other responses had been tested, the plateau counter was set to zero to reflect the age of the response. The end-game aspect and nose angle indicated that the agent was almost directly in front of the target aircraft, separated by 128 m. At the end of the simulation the agent's velocity was still 150 m/s greater than the target, a factor contributing to the agent fly-by past the target and loss of a final shot advantage. Nevertheless, during the simulation the agent was able to accumulate 9 seconds of shot time, none of which was estimated. Since the simulation was terminated the instant NOSTRUM detected a fly-by there was no opportunity for any time in danger to accumulate. Finally, since this was the first response explored, and NOSTRUM had no others to compare it to, the interestingness of the default response was 0.

Although presenting the data in tabular fashion is sometimes useful, it is often easier to interpret simulation results graphically. Figure 5.2 provides two- and three-dimensional views of agent behavior relative to the target using the default response. It is apparent when viewing the data graphically that the agent stayed at nearly the same altitude as the target throughout the simulation, even though it might have reduced its airspeed and held a shot position for much longer through a climb or deceleration. This is clearly a case when proportional control is not the most desirable response.

							END-GAME POSITION						TIMES			
LC	PC	((NNNNN	WWWWW	TTTTT)	AAAA)	ASP	NOS	VDEL	RANGE	FSH	FBY	ST	EST	DT	INT
1	0	((0	0	0)	0.0)	174	174	150	128	NIL	T	9	0	0	0
2	1	((0	0	500)	0.0)	176	99	141	508	NIL	T	9	0	0	0
3	2	((-500	0	0)	0.0)	129	129	150	89	NIL	T	9	0	0	0
4	3	((0	0	0)	-0.2)	125	125	109	87	NIL	T	9	0	0	0
5	0	((0	0	1000)	0.0)	168	91	134	1003	NIL	T	11	0	0	35
6	1	((0	0	1500)	0.0)	101	90	125	1551	NIL	T	10	0	0	35
7	2	((-500	0	1000)	0.0)	154	93	134	984	NIL	T	11	0	0	35
8	3	((0	0	1000)	-0.2)	2	72	83	1063	NIL	NIL	10	0	0	44
9	4	((0	0	1000)	-0.4)	2	36	27	1785	NIL	NIL	11	0	0	46
10	5	((0	0	1000)	-0.6)	1	14	1	3078	NIL	NIL	11	0	0	55
11	6	((0	0	1000)	-0.8)	0	8	-1	3873	NIL	NIL	11	0	0	59
12	0	((0	0	1000)	-1.0)	0	7	0	4368	T	NIL	999	999	0	106
13	1	((0	0	500)	-1.0)	0	3	0	3727	NIL	NIL	11	0	0	108
14	2	((-500	0	500)	-1.0)	0	3	-1	3971	NIL	NIL	11	0	0	108
15	0	((-500	0	1000)	-1.0)	0	9	-1	4539	T	NIL	999	999	0	153
16	1	((-500	0	1000)	-0.8)	0	9	-2	3989	NIL	NIL	12	0	0	153
17	2	((-500	0	1500)	-0.8)	0	13	2	4351	T	NIL	125	108	0	163
18	3	((0	0	1500)	-0.8)	0	11	1	4384	T	NIL	176	156	0	163
19	4	((-500	0	1500)	-0.6)	0	20	-2	3476	NIL	NIL	8	0	0	163
20	5	((500	0	1500)	-0.8)	0	11	1	4401	T	NIL	166	147	0	163
21	6	((0	0	1500)	-0.6)	0	20	-1	3505	NIL	NIL	10	0	0	163
22	7	((-500	0	2000)	-0.6)	0	22	2	3670	NIL	NIL	4	0	0	163
23	8	((-1000	0	1500)	-0.6)	0	20	0	3299	NIL	NIL	4	0	0	163
24	9	((500	0	1000)	-0.8)	0	8	-1	4028	T	NIL	999	999	0	163
25	10	((1000	0	1500)	-0.8)	0	10	1	4373	T	NIL	149	129	0	163
26	11	((500	0	1500)	-0.6)	0	18	-2	3561	NIL	NIL	11	0	0	163
27	12	((0	0	2000)	-0.6)	0	22	-1	3957	NIL	NIL	5	0	0	163
28	13	((-500	0	2000)	-0.8)	0	16	-2	4547	NIL	NIL	13	0	0	163
29	14	((-500	0	2500)	-0.6)	0	29	36	4310	NIL	NIL	15	0	0	163
30	15	((-1000	0	2000)	-0.6)	0	30	-1	3626	NIL	NIL	4	0	0	163
31	16	((-1500	0	1500)	-0.6)	0	19	1	3329	NIL	NIL	4	0	0	163
32	17	((-1000	0	1500)	-0.8)	0	14	-2	4303	T	NIL	999	999	0	173
33	18	((-1000	0	1000)	-0.8)	0	9	1	3780	NIL	NIL	14	0	0	173
34	19	((-1000	0	1000)	-1.0)	0	8	0	4490	T	NIL	999	999	0	183
35	20	((-1000	0	500)	-1.0)	0	3	-2	3944	NIL	NIL	11	0	0	183

Table 5.3 **The complete learning cycle for plan sector 0-0-0-0-0.**

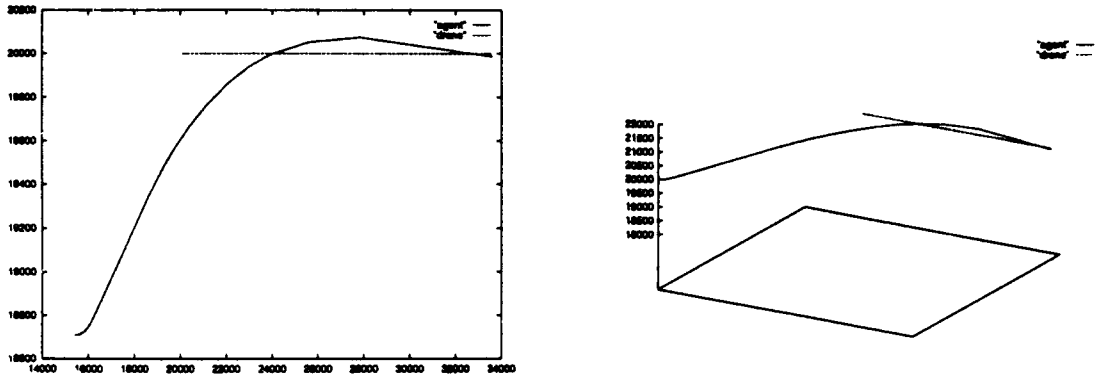

Figure 5.2 **Default agent behavior in plan sector 0-0-0-0-0.** Two- and three-dimensional results.

Starting with the default response, NOSTRUM began suggesting changes and tested a variety of other responses, as show in Table 5.3. NOSTRUM noticed immediately that the agent was flying past the target and suggested several modifications to prevent this from happening. The response heuristics suggest climbing above the target, lagging the target, or decelerating when a fly-by occurs. Each modification was tested in turn during learning cycles 2, 3 and 4, but none of these was able to prevent the agent from flying beyond the target. At learning cycle 5, however, NOSTRUM further increased the altitude of the fly-to point over the target and recorded an increase in interestingness because the response improved the shot time and was the best response explored so far. The jump in interestingness was attributed to the application of response heuristic **RH4**, which received the 35 point bonus and was re-applied again during cycle 6. Increasing the fly-to point altitude failed to improve the response, and interestingness did not increase again until NOSTRUM tried combining deceleration with climbing at cycle 8.

Although no immediate increase in shot time occurred, NOSTRUM repeatedly applied RH6 in cycles 8 through 12 because each application resulted in a slight interestingness increase. As a result, NOSTRUM discovered a better response at cycle 12 that produced a

significant increase in shot time[1]. From cycle 12 on, NOSTRUM tried to improve upon the response further using various combinations of power settings, lag pursuit, and climbing above the target to bleed off excess airspeed. At cycle 15 the best response was actually detected by adding a slight lag pursuit to the response explored during cycle 12. Additional exploration failed to improve the agent shot time, and the plateau counter stopped additional exploration at cycle 35. Figure 5.3 graphically shows all of the responses explored by NOSTRUM, and Figure 5.4 shows the best response explored for plan sector 0-0-0-0-0.

The best response explored for plan sector 0-0-0-0-0 was `((-500 0 1000) -1.0)`. Interpreted, the response directed the agent to lag the target by 500 m, climb above the target by 1000 m, and simultaneously reduce power by 100%. This is close to what might be intuitively expected. The starting velocity of the agent was quite a bit greater than that of the target and, left unchecked, the agent was doomed to fly by the target. NOSTRUM found early on that climbing increased its shot time, but this alone was not enough. Soon, it discovered that combining deceleration with climbing not only increased the shot time but prevented a fly-by as well. When the power setting had been decreased as much as possible, NOSTRUM then found utility in slightly lagging the target.

5.2.1.2 Plan Sector 1-0-0-0-0 Results. Plan sector 1-0-0-0-0, the second one explored by NOSTRUM, is also a pure plan sector. The initial training conditions for sector 1-0-0-0-0, shown in Table 5.4, are identical to those for sector 0-0-0-0-0 except that the agent is moving slower than the target.

Since the agent was traveling slower than the target, it was impossible for the agent to fly by the target using the default response. Although the agent was able to launch a missile in the end-game position, NOSTRUM tried to improve the response by extending the duration of the shot time window. Table 5.5 shows the complete learning cycle for plan sector 1-0-0-0-0.

NOSTRUM discovered early in the learning cycle that applying response heuristic **RH3** produced positive results. Consequently, **RH3** received a bonus and was applied first for several learning cycles starting with cycle 4. At cycle 8, NOSTRUM had increased the

[1]For aesthetic reasons, shot times that are greater than 999 seconds are displayed in the table as "999."

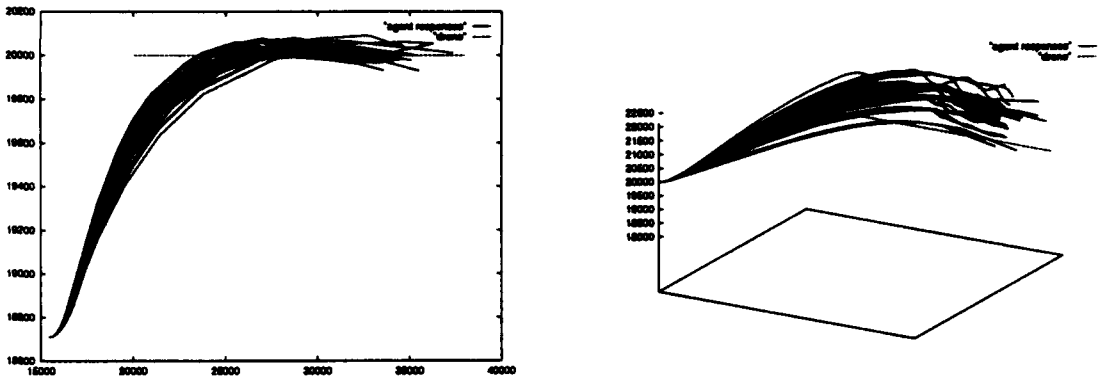

Figure 5.3 **All responses tested for plan sector 0-0-0-0-0.** Two- and three-dimensional results.

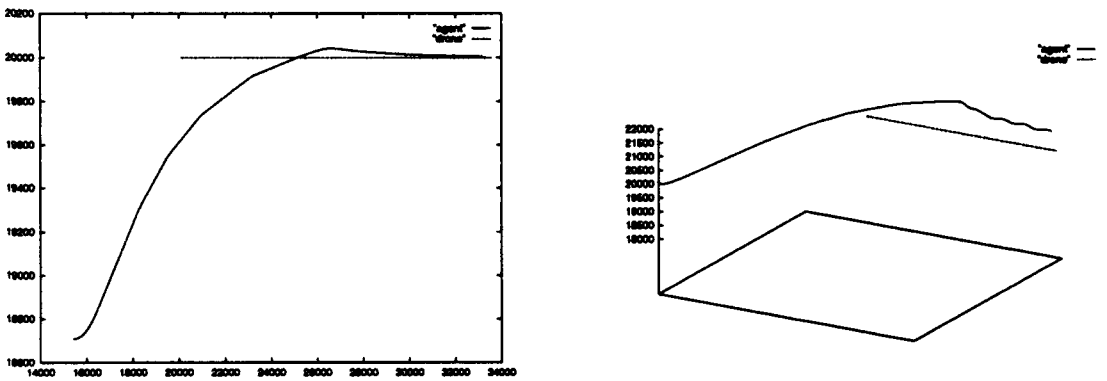

Figure 5.4 **Best response found for plan sector 0-0-0-0-0.** Two- and three-dimensional results.

Training Scenario for Sector 1-0-0-0-0		
Identifier Digit Position	Value	Meaning
1	1	Agent velocity is 150 m/s slower than target
2	0	Agent heading is parallel to target
3	0	Agent range to target is 5000 m
4	0	Agent altitude is equal to target's
5	0	Agent aspect angle to target is 15 degrees

Table 5.4 **Training scenario created for plan sector 1-0-0-0-0.**

power setting to its maximum value and began exploring other ways to improve the shot time. The system recognized that additional progress might be made by shrinking the end-game velocity delta. Since the target was still moving faster than the agent, NOSTRUM suggested a slight dive beneath the target in order to increase agent acceleration. Later during the learning cycle, when adjustments to set the fly-to point further beneath the target no longer had a positive impact, NOSTRUM then tried combining lead pursuit with the best response. At learning cycle 30, NOSTRUM discovered the best response explored, combining acceleration, diving, and lead pursuit into the plan sector response. Like the results produced for plan sector 0-0-0-0-0, this is what we intuitively might expect. The responses explored by NOSTRUM are displayed graphically in Figures 5.5 through Figure 5.7.

An interesting event occurred between learning cycles 9 and 10 for the plan sector 1-0-0-0-0 response that is somewhat counter-intuitive. Looking at cycle 9 in Table 5.5, we can see that the final velocity delta between agent and target was -93 m/s. The negative sign indicates that the target was moving faster than the agent. We might expect that if the agent dove more sharply to increase airspeed, as NOSTRUM suggested in learning cycle 10, that the final velocity delta would actually be less than before. The simulation results, however, indicate that increasing the dive angle had the *opposite* effect, and that the final velocity delta was actually worse than it was in cycle 8 when the agent didn't dive at all.

It has been said that DBL systems are trapped in the world defined by the simulator (20). I suspect that the program output presented here illustrates an occurrence of the system abandoning a path in the search tree because the simulator did not accurately represent the real world. If a higher fidelity simulator had been used it is likely that NOS-

							END-GAME POSITION						TIMES			
LC	PC	((NNNNN	WWWWW	TTTTT)	AAAA)	ASP	NOS	VDEL	RANGE	FSH	FBY	ST	EST	DT	INT
1	0	((0	0	0)	0.0)	1	1	-150	12569	T	NIL	78	25	0	0
2	1	((0	0	-500)	0.0)	1	1	-149	12481	T	NIL	78	25	0	0
3	2	((500	0	0)	0.0)	1	1	-150	12566	T	NIL	78	25	0	0
4	0	((0	0	0)	0.2)	1	1	-141	12181	T	NIL	81	28	0	35
5	0	((0	0	0)	0.4)	1	1	-132	11755	T	NIL	84	31	0	70
6	0	((0	0	0)	0.6)	1	1	-123	11348	T	NIL	88	35	0	105
7	0	((0	0	0)	0.8)	0	0	-110	10789	T	NIL	95	42	0	149
8	0	((0	0	0)	1.0)	0	0	-100	10344	T	NIL	101	48	0	184
9	0	((0	0	-500)	1.0)	0	0	-93	9894	T	NIL	108	55	0	219
10	1	((0	0	-1000)	1.0)	0	0	-105	10478	T	NIL	98	46	0	219
11	2	((500	0	-500)	1.0)	0	0	-96	10027	T	NIL	105	52	0	219
12	3	((0	0	-1500)	1.0)	0	0	-113	10887	T	NIL	94	40	0	219
13	4	((500	0	-1000)	1.0)	0	0	-103	10429	T	NIL	99	46	0	219
14	5	((1000	0	-500)	1.0)	0	0	-96	10089	T	NIL	105	52	0	219
15	6	((0	0	-2000)	1.0)	0	0	-115	10850	T	NIL	93	40	0	219
16	7	((500	0	-1500)	1.0)	0	0	-102	10328	T	NIL	101	48	0	219
17	8	((1000	0	-1000)	1.0)	0	0	-92	9880	T	NIL	108	55	0	219
18	9	((1500	0	-500)	1.0)	0	0	-98	10108	T	NIL	104	51	0	219
19	10	((0	0	-2500)	1.0)	1	1	-125	11374	T	NIL	84	35	0	219
20	11	((500	0	-2000)	1.0)	0	0	-114	10829	T	NIL	92	40	0	219
21	12	((1000	0	-1500)	1.0)	0	0	-109	10588	T	NIL	96	43	0	219
22	13	((1500	0	-1000)	1.0)	0	0	-92	9893	T	NIL	108	55	0	219
23	14	((2000	0	-500)	1.0)	0	0	-92	9890	T	NIL	108	55	0	219
24	15	((0	0	-3000)	1.0)	1	1	-123	11295	T	NIL	84	35	0	219
25	16	((500	0	-2500)	1.0)	0	0	-111	10760	T	NIL	92	41	0	219
26	17	((1000	0	-2000)	1.0)	0	0	-113	10824	T	NIL	94	41	0	219
27	18	((1500	0	-1500)	1.0)	0	0	-104	10390	T	NIL	99	46	0	219
28	19	((2000	0	-1000)	1.0)	0	0	-92	9892	T	NIL	108	55	0	219
29	0	((2500	0	-500)	1.0)	0	0	-91	9870	T	NIL	109	56	0	254
30	0	((3000	0	-500)	1.0)	0	0	-83	9489	T	NIL	116	63	0	289
31	1	((3500	0	-500)	1.0)	0	0	-83	9488	T	NIL	116	63	0	289
32	2	((3000	0	-1000)	1.0)	0	0	-92	9869	T	NIL	108	55	0	289
33	3	((3500	0	-1000)	1.0)	0	0	-91	9839	T	NIL	109	56	0	289
34	4	((4000	0	-500)	1.0)	0	0	-83	9485	T	NIL	116	63	0	289
35	5	((3000	0	-1500)	1.0)	0	0	-92	9880	T	NIL	108	55	0	289
36	6	((3500	0	-1500)	1.0)	0	0	-92	9879	T	NIL	108	55	0	289
37	7	((4000	0	-1000)	1.0)	0	0	-91	9791	T	NIL	109	56	0	289
38	8	((4500	0	-500)	1.0)	0	0	-83	9484	T	NIL	116	63	0	289
39	9	((3000	0	-2000)	1.0)	0	0	-104	10351	T	NIL	99	46	0	289
40	10	((3500	0	-2000)	1.0)	0	0	-98	10104	T	NIL	103	50	0	289
41	11	((4000	0	-1500)	1.0)	0	0	-87	9603	T	NIL	112	60	0	289
42	12	((4500	0	-1000)	1.0)	0	0	-90	9771	T	NIL	110	57	0	289
43	13	((5000	0	-500)	1.0)	0	0	-83	9483	T	NIL	116	63	0	289
44	14	((3000	0	-2500)	1.0)	0	0	-100	10222	T	NIL	102	49	0	289
45	15	((3500	0	-2500)	1.0)	0	0	-100	10219	T	NIL	102	49	0	289
46	16	((4000	0	-2000)	1.0)	0	0	-94	9972	T	NIL	106	53	0	289
47	17	((4500	0	-1500)	1.0)	0	0	-92	9878	T	NIL	108	55	0	289
48	18	((5000	0	-1000)	1.0)	0	0	-86	9561	T	NIL	114	61	0	289
49	19	((5500	0	-500)	1.0)	0	0	-83	9478	T	NIL	116	63	0	289
50	20	((3000	0	-3000)	1.0)	1	1	-114	10832	T	NIL	92	40	0	289

Table 5.5 **The complete learning cycle for plan sector 1-0-0-0-0.**

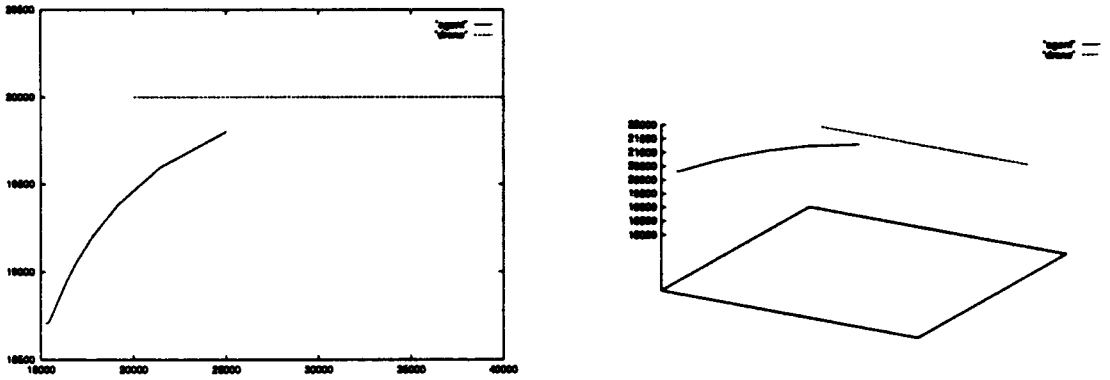

Figure 5.5 **Default agent behavior in plan sector 1-0-0-0-0.** Two- and three-dimensional results.

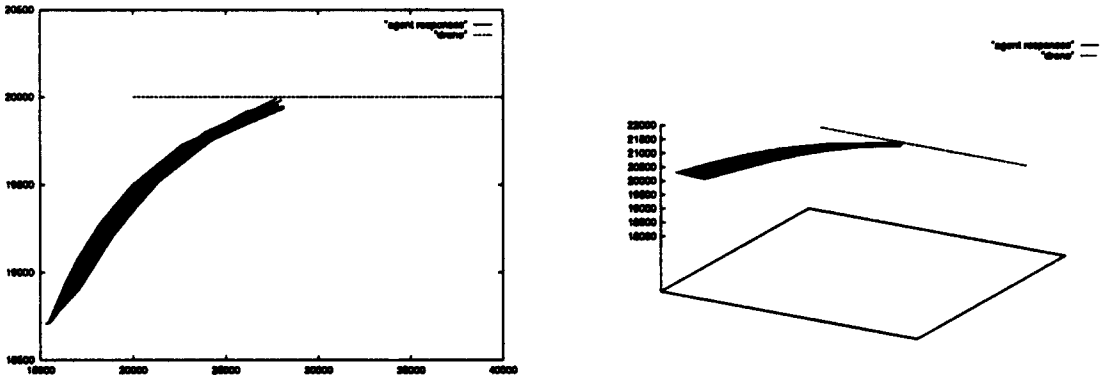

Figure 5.6 **All responses tested for plan sector 1-0-0-0-0.** Two- and three-dimensional results.

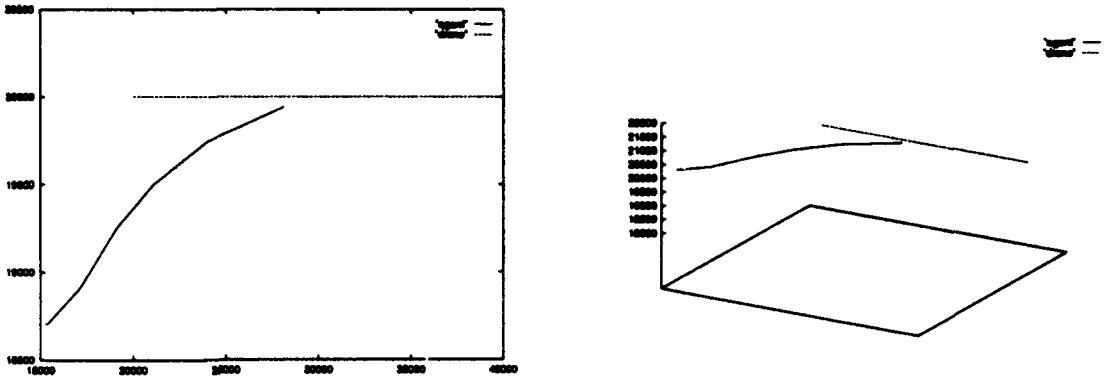

Figure 5.7 **Best response found for plan sector 1-0-0-0-0.** Two- and three-dimensional results.

TRUM would have continued diving to close the gap in velocities until it started having a negative impact on the final shot position.

5.2.2 Learning in Dirty Sectors. NOSTRUM was able to discover better responses in pure plan sectors more readily than in the "dirty" sectors (where many boundaries were crossed) because credit assignment was straightforward: the performance of the agent could be attributed entirely to the response currently being explored. As NOSTRUM moved away from the pure sectors and towards dirtier ones credit assignment was a more difficult issue to resolve. As an example, consider the results produced by NOSTRUM in plan sector 0-0-0-0-3. The initial conditions for this training scenario are shown in Table 5.6, and are also shown graphically in Figure 5.8.

NOSTRUM did not have much success improving the response for this plan sector. Although NOSTRUM more than doubled the agent's shot time, it began to explore a dead-end path and never recovered. As shown in Table 5.7, NOSTRUM discovered at learning cycle 16 that adding an offset to the fly-to point off the target wing was reducing the end-game aspect angle, causing an increase in interestingness. A shrinking aspect angle may be an indication that the agent is moving more towards a position behind the target. In this plan sector, however, the decrease in aspect angle was coupled with a decrease

Training Scenario for Sector 0-0-0-0-3		
Identifier Digit Position	Value	Meaning
1	0	Agent velocity is 150 m/s greater than target
2	0	Agent heading is parallel to target
3	0	Agent range to target is 5000 m
4	0	Agent altitude is equal to target's
5	3	Agent aspect angle to target is 105 degrees

Table 5.6 **Training scenario created for plan sector 0-0-0-0-3.**

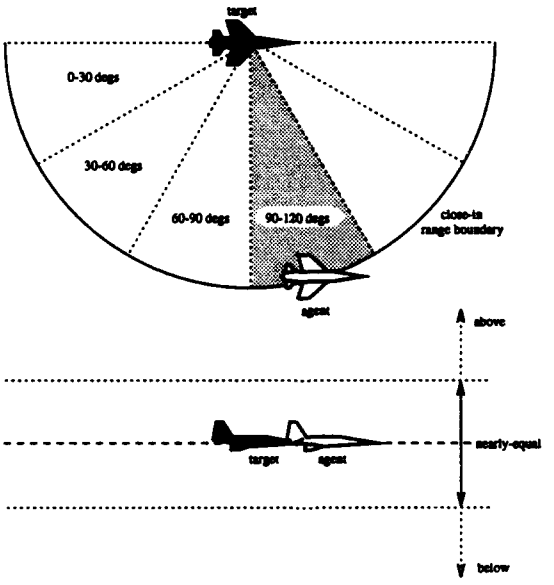

Figure 5.8 **A dirty plan sector.** Training scenario for plan sector, 0-0-0-0-3, depicted graphically.

in end-game range: the agent was now too close to fire upon the target in the end-game position.

			END-GAME POSITION						TIMES			
LC	PC	((XXXXX YYYYY TTTTT) AAAA)	ASP	NOS	VDEL	RANGE	FSH	FBY	ST	EST	DT	INT
1	0	((0 0 0) 0.0)	18	31	152	5292	T	NIL	5	4	0	0
2	0	((0 0 -500) 0.0)	13	30	156	5440	T	NIL	8	5	0	39
3	1	((0 0 -1000) 0.0)	16	28	159	5617	T	NIL	7	5	0	41
4	2	((0 0 -1500) 0.0)	16	27	160	5410	T	NIL	7	4	0	43
5	0	((0 0 -2000) 0.0)	11	26	158	5517	T	NIL	9	5	0	82
6	1	((0 0 -2500) 0.0)	8	26	154	5484	T	NIL	9	5	0	84
7	2	((0 0 -3000) 0.0)	6	26	138	5576	T	NIL	9	6	0	86
8	0	((0 0 -3500) 0.0)	7	26	138	5255	T	NIL	13	5	0	121
9	1	((0 0 -4000) 0.0)	7	24	136	5101	T	NIL	12	4	0	123
10	2	((0 0 -4500) 0.0)	6	22	132	4652	T	NIL	10	2	0	125
11	3	((0 0 -5000) 0.0)	5	22	129	4399	T	NIL	12	2	0	127
12	4	((0 0 -5500) 0.0)	5	19	124	4340	T	NIL	12	1	0	129
13	5	((0 0 -6000) 0.0)	5	20	125	4098	T	NIL	10	0	0	129
14	6	((500 0 -5500) 0.0)	6	19	124	4310	T	NIL	12	1	0	129
15	7	((0 0 -5500) 0.2)	5	19	129	4453	T	NIL	12	2	0	129
16	8	((0 500 -5500) 0.0)	4	19	124	4000	T	NIL	11	0	0	131
17	9	((0 1000 -5500) 0.0)	5	18	115	3798	NIL	NIL	8	0	0	133
18	10	((0 1500 -5500) 0.0)	3	18	99	3206	NIL	NIL	0	0	0	135
19	11	((0 2000 -5500) 0.0)	2	16	68	3078	NIL	NIL	0	0	0	144
20	12	((0 2500 -5500) 0.0)	1	19	64	2806	NIL	NIL	0	0	0	146
21	13	((0 3000 -5500) 0.0)	1	16	56	3074	NIL	NIL	0	0	0	146
22	14	((0 2500 -5000) 0.0)	1	19	67	2860	NIL	NIL	0	0	0	146
23	15	((-500 2500 -5500) 0.0)	1	19	64	2803	NIL	NIL	0	0	0	146
24	16	((0 2500 -5500) -0.2)	1	19	62	2825	NIL	NIL	0	0	0	146
25	17	((0 3000 -5000) 0.0)	1	18	59	2872	NIL	NIL	0	0	0	146
26	18	((-500 3000 -5500) 0.0)	1	17	58	2979	NIL	NIL	0	0	0	146
27	19	((0 3000 -5500) -0.2)	1	17	52	2986	NIL	NIL	0	0	0	146
28	20	((0 3500 -5500) 0.0)	1	15	52	3091	NIL	NIL	0	0	0	148

Table 5.7 **The complete learning cycle for plan sector 0-0-0-0-3.**

For many of the plan sectors, NOSTRUM ran into similar dead-ends. Much of the problem was tied to the sets of interestingness and response heuristics. The heuristics needed to be general enough to accommodate the multitude of situations that NOSTRUM would encounter during discovery, but developing such heuristics turned out to be an incredibly complex task. Consequently, the heuristics that worked superbly in sector 0-0-0-0-0 failed to have the same success in many other sectors. This was especially true in the opposing heading crossing angle sectors. Refer to Table 5.8 for an example of a learning cycle where

the heading-crossing index was 1, indicating that the agent was traveling in the opposite direction as the target. The suggestions made by NOSTRUM had very little impact on the total accumulated shot time, and in the end NOSTRUM was unable to find a response that improved upon the default response.

			END-GAME POSITION						TIMES			
LC	PC	((HHHHH VVVVV TTTTT) AAAA)	ASP	HOS	VDEL	RANGE	FSH	FBY	ST	EST	DT	INT
1	0	((0 0 0) 0.0)	3	6	111	10823	T	NIL	57	31	0	0
2	1	((0 0 -500) 0.0)	3	6	110	10789	T	NIL	57	31	0	0
3	2	((500 0 0) 0.0)	3	6	111	10822	T	NIL	57	31	0	0
4	3	((0 0 0) 0.2)	3	7	113	10815	T	NIL	56	30	0	0
5	4	((0 0 -1000) 0.0)	3	6	111	10847	T	NIL	57	31	0	0
6	5	((500 0 -500) 0.0)	3	6	110	10789	T	NIL	57	31	0	0
7	6	((0 0 -500) 0.2)	3	7	112	10827	T	NIL	56	30	0	0
8	7	((1000 0 0) 0.0)	3	6	111	10822	T	NIL	57	31	0	0
9	8	((500 0 0) 0.2)	3	7	113	10815	T	NIL	56	30	0	0
10	9	((0 0 0) 0.4)	3	7	114	10806	T	NIL	56	30	0	0
11	10	((0 0 -1500) 0.0)	3	6	111	10854	T	NIL	57	31	0	0
12	11	((500 0 -1000) 0.0)	3	6	111	10847	T	NIL	57	31	0	0
13	12	((0 0 -1000) 0.2)	3	7	112	10837	T	NIL	56	30	0	0
14	13	((1000 0 -500) 0.0)	3	6	110	10788	T	NIL	57	31	0	0
15	14	((500 0 -500) 0.2)	3	7	112	10827	T	NIL	56	30	0	0
16	15	((0 0 -500) 0.4)	3	7	114	10818	T	NIL	56	30	0	0
17	16	((1500 0 0) 0.0)	3	6	111	10822	T	NIL	57	31	0	0
18	17	((1000 0 0) 0.2)	3	7	113	10815	T	NIL	56	30	0	0
19	18	((500 0 0) 0.4)	3	7	114	10806	T	NIL	56	30	0	0
20	19	((0 0 0) 0.6)	3	7	115	10794	T	NIL	56	30	0	0
21	20	((0 0 -2000) 0.0)	3	6	111	10861	T	NIL	57	31	0	0

Table 5.8 **The complete learning cycle for plan sector 0-1-0-1-0.** An opposing heading-crossing angle sector.

The primary reason that responses in the opposing HCA sectors had such minimal impact is that the agent spent only a short period of time in each one of those sectors. There were two factors contributing to the reduction in time. First, since the agent and target were traveling along heading vectors that were parallel but in opposite directions, it was only a matter of a few seconds before the agent moved out of the training sector and into an adjacent sector. This action is shown in Figure 5.9. At time *t1*, the agent and target were both in their starting positions for the training scenario. Moments later, at time *t2*, the relative positions of the aircraft had changed such that the agent was no

longer in the training scenario sector but in an adjacent sector. Secondly, MAXIM agents enter the simulation in search mode and it takes some number of simulation cycles for the agent to recognize and lock in on a target aircraft. By the time the agent selected the target aircraft it is quite possible that enough time had elapsed to place the agent in the situation at time *t2*. Together, these factors frequently made it difficult to improve the responses in opposing HCA sectors.

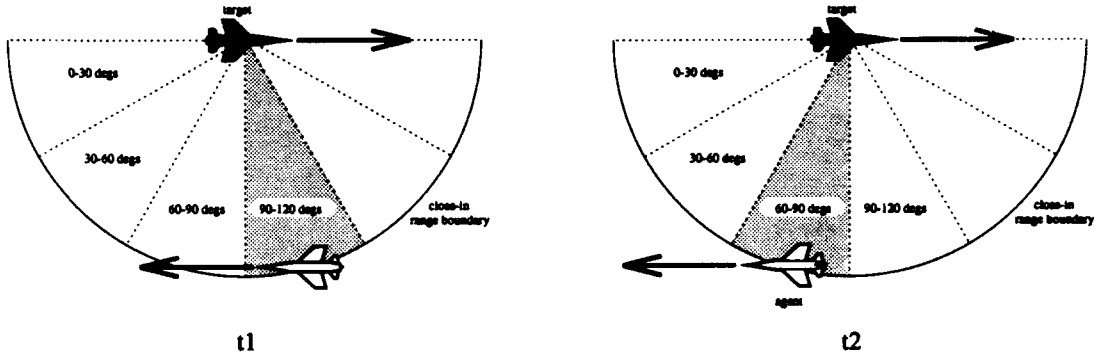

Figure 5.9 **Sector boundary crossings.** Rapid changes in opposing heading-crossing angles sectors made it difficult for these responses to have a significant effect.

5.2.3 Overall Learning Success. The previous section presented data collected from NOSTRUM's exploration of a select few plan sectors. This should not suggest, however, that NOSTRUM was not able to improve the responses of the other 141 plan sectors. Tables 5.9 and 5.10 present data collected from all plan sectors. The format of the tables permits comparison of the shot and danger times for a standard agent (indicated in the table as *DEFAULT*) to that of a flexible agent after learning (indicated in the table as *AFTER*), and the increase or decrease resulting from learning (indicated in the table as *DELTA*).

In 96 out of the 144 training scenarios, NOSTRUM was able to improve the total shot time compared to the time available to a proportionally controlled agent. In many plan sectors the improvement was quite significant, giving the agent virtually unlimited shot

Results of Learning - Close-Range Plan Sectors												
PLAN SECTOR	SCENARIO STARTING CONDITIONS					DEFAULT		RESPONSE LEARNED	AFTER		DELTA	
	VDEL	HCA	RNG	ALT	ASP	ST	DT		ST	DT	ST	DT
0-0-0-0-0	150	P	5000	0	15	9	0	((-500 0 1000) -1.0)	15325	0	15310	0
1-0-0-0-0	-150	P	5000	0	15	78	0	((3000 0 -500) 1.0)	116	0	38	0
0-1-0-0-0	150	O	5000	0	15	51	0	((0 0 -2500) 0.0)	62	0	11	0
1-1-0-0-0	-150	O	5000	0	15	49	0	((0 0 0) 0.6)	50	0	1	0
0-0-0-1-0	150	P	5000	2500	15	2	0	((0 0 1500) -0.2)	8	0	6	0
1-0-0-1-0	-150	P	5000	2500	15	72	0	((4500 0 -1000) 1.0)	5925	0	8057	0
0-1-0-1-0	150	O	5000	2500	15	57	0	((0 0 0) 0.0)	57	0	0	0
1-1-0-1-0	-150	O	5000	2500	15	57	0	((0 0 0) 0.6)	58	0	1	0
0-0-0-2-0	150	P	5000	-2500	15	2	0	((500 0 500) 0.0)	9648	0	9646	0
1-0-0-2-0	-150	P	5000	-2500	15	52	0	((1500 0 -2000) 1.0)	77	0	25	0
0-1-0-2-0	150	O	5000	-2500	15	42	0	((0 0 -4000) 0.0)	46	0	4	0
1-1-0-2-0	-150	O	5000	-2500	15	47	0	((0 0 -1000) 0.0)	48	0	1	0
0-0-0-0-1	150	P	5000	0	45	0	0	((-1500 0 6500) -1.0)	17	0	17	0
1-0-0-0-1	-150	P	5000	0	45	68	0	((7500 0 -1000) 1.0)	308	0	240	0
0-1-0-0-1	150	O	5000	0	45	47	0	((0 0 -3500) 0.0)	51	0	4	0
1-1-0-0-1	-150	O	5000	0	45	56	0	((0 0 0) 0.0)	56	0	0	0
0-0-0-1-1	150	P	5000	2500	45	0	0	((0 0 0) 0.0)	0	0	0	0
1-0-0-1-1	-150	P	5000	2500	45	101	0	((2000 0 -1000) 0.6)	3504	0	3403	0
0-1-0-1-1	150	O	5000	2500	45	51	0	((0 0 0) 0.0)	51	0	0	0
1-1-0-1-1	-150	O	5000	2500	45	65	0	((0 0 0) 0.2)	66	0	1	0
0-0-0-2-1	150	P	5000	-2500	45	0	0	((0 0 4000) 0.0)	14	0	14	0
1-0-0-2-1	-150	P	5000	-2500	45	50	0	((4000 0 -2500) 1.0)	128	0	78	0
0-1-0-2-1	150	O	5000	-2500	45	42	0	((0 0 -6000) 0.0)	47	0	5	0
1-1-0-2-1	-150	O	5000	-2500	45	53	0	((0 0 0) 0.0)	53	0	0	0
0-0-0-0-2	150	P	5000	0	75	0	0	((-500 0 1500) 0.0)	4	0	4	0
1-0-0-0-2	-150	P	5000	0	75	60	0	((4500 0 -2000) 1.0)	115324	0	115264	0
0-1-0-0-2	150	O	5000	0	75	39	0	((0 0 -2500) 0.2)	44	0	5	0
1-1-0-0-2	-150	O	5000	0	75	58	0	((0 0 0) 0.2)	59	0	1	0
0-0-0-1-2	150	P	5000	2500	75	0	0	((0 0 0) 0.2)	0	0	0	0
1-0-0-1-2	-150	P	5000	2500	75	97	0	((2500 0 -1500) 0.2)	21471431	0	21471334	0
0-1-0-1-2	150	O	5000	2500	75	40	0	((0 0 0) 0.0)	40	0	0	0
1-1-0-1-2	-150	O	5000	2500	75	72	0	((0 0 -500) 0.2)	73	0	1	0
0-0-0-2-2	150	P	5000	-2500	75	0	0	((0 0 3000) -0.4)	8	0	8	0
1-0-0-2-2	-150	P	5000	-2500	75	46	0	((1500 0 -1500) 0.8)	909	0	863	0
0-1-0-2-2	150	O	5000	-2500	75	35	0	((0 0 -3000) 0.0)	44	0	9	0
1-1-0-2-2	-150	O	5000	-2500	75	51	0	((0 0 -500 0.6)	53	0	2	0
0-0-0-0-3	150	P	5000	0	105	0	0	((0 0 -3500) 0.0)	13	0	13	0
1-0-0-0-3	-150	P	5000	0	105	64	0	((0 0 -500) 0.0)	33648	0	33584	0
0-1-0-0-3	150	O	5000	0	105	30	0	((0 0 0) 0.0)	35	0	5	0
1-1-0-0-3	-150	O	5000	0	105	62	0	((0 0 0) 1.0)	100	0	38	0
0-0-0-1-3	150	P	5000	2500	105	0	0	((0 4000 500) 0.0)	11	0	11	0
1-0-0-1-3	-150	P	5000	2500	105	102	0	((0 500 -500) 0.0)	63	0	-39	0
0-1-0-1-3	150	O	5000	2500	105	32	0	((0 0 -500) 0.0)	35	0	3	0
1-1-0-1-3	-150	O	5000	2500	105	81	0	((0 0 -500) 0.6)	95	0	14	0
0-0-0-2-3	150	P	5000	-2500	105	0	0	((500 1500 -4000) 0.0)	17	0	17	0
1-0-0-2-3	-150	P	5000	-2500	105	48	0	((-500 0 -500) 1.0)	616	0	568	0
0-1-0-2-3	150	O	5000	-2500	105	31	0	((0 0 -500) 0.0)	38	0	7	0
1-1-0-2-3	-150	O	5000	-2500	105	47	0	((0 500 0) 0.2)	62	0	15	0
0-0-0-0-4	150	P	5000	0	135	34	0	((0 0 0) 0 0)	0	0	-34	0
1-0-0-0-4	-150	P	5000	0	135	65	0	((-5500 0 -2500) 0.0)	99257	0	99192	0
0-1-0-0-4	150	O	5000	0	135	21	0	((0 0 0) 0.0)	25	0	4	0
1-1-0-0-4	-150	O	5000	0	135	70	0	((1000 0 -1000) 0.2)	22444	0	22374	0
0-0-0-1-4	150	P	5000	2500	135	0	0	((0 1500 4500) 0.0)	8	0	8	0
1-0-0-1-4	-150	P	5000	2500	135	89	0	((0 0 0) 0.0)	0	0	-89	0
0-1-0-1-4	150	O	5000	2500	135	22	0	((0 0 -1000) 0.0)	45	0	23	0
1-1-0-1-4	-150	O	5000	2500	135	97	0	((2500 -500 -1500) 0.2)	23724	0	23627	0
0-0-0-2-4	150	P	5000	-2500	135	34	0	((0 -3000 2000) 0.0)	1	0	-33	0
1-0-0-2-4	-150	P	5000	-2500	135	52	0	((1000 0 -500) 1.0)	3464	0	3412	0
0-1-0-2-4	150	O	5000	-2500	135	21	0	((0 0 0) 0.2)	30	0	9	0
1-1-0-2-4	-150	O	5000	-2500	135	54	0	((0 1000 -1000) 0.2)	380	0	326	0
0-0-0-0-5	150	P	5000	0	165	31	4	((0 0 -2000) 0.0)	34	4	3	0
1-0-0-0-5	-150	P	5000	0	165	54	3	((0 0 -1500) 0.0)	444	3	390	0
0-1-0-0-5	150	O	5000	0	165	3	1	((0 0 500) 0.0)	18	1	15	0
1-1-0-0-5	-150	O	5000	0	165	69	2	((0 0 -3000) 0.0)	385	2	316	0
0-0-0-1-5	150	P	5000	2500	165	32	0	((0 0 0) 0.0)	0	0	-32	0
1-0-0-1-5	-150	P	5000	2500	165	66	0	((0 0 0) 0.0)	0	0	-66	0
0-1-0-1-5	150	O	5000	2500	165	10	0	((0 0 -500) 0.0)	27	0	17	0
1-1-0-1-5	-150	O	5000	2500	165	85	0	((1000 500 2000) 0.0)	20637	0	20552	0
0-0-0-2-5	150	P	5000	-2500	165	32	0	((0 500 0) 0.0)	32	0	0	0
1-0-0-2-5	-150	P	5000	-2500	165	58	0	((500 1500 -1000) 0.6)	25399	0	25341	0
0-1-0-2-5	150	O	5000	-2500	165	3	0	((500 500 -500) 0.0)	19	0	16	0
1-1-0-2-5	-150	O	5000	-2500	165	52	0	((1500 500 -1500) 1.0)	388	0	336	0

Table 5.9 **Results of learning in close range plan sectors.**

Results of Learning - Far-Away Plan Sectors												
PLAN SECTOR	SCENARIO STARTING CONDITIONS					DEFAULT		RESPONSE LEARNED	AFTER		DELTA	
	VDEL	HCA	RNG	ALT	ASP	ST	DT		ST	DT	ST	DT
0-0-1-0-0	150	P	15000	0	15	82	0	((500 0 -2000) 0.0)	176037	0	175955	0
1-0-1-0-0	-150	P	15000	0	15	35	0	((0 0 0) 0.0)	35	0	0	0
0-1-1-0-0	150	O	15000	0	15	78	0	((0 0 -10000) 0.2)	129	0	51	0
1-1-1-0-0	-150	O	15000	0	15	0	0	((0 0 0) 0.0)	0	0	0	0
0-0-1-1-0	150	P	15000	7500	15	39	0	((4500 0 -3500) -1.0)	216363	0	216324	0
1-0-1-1-0	-150	P	15000	7500	15	191	0	((0 0 0) 0.0)	155	0	-36	0
0-1-1-1-0	150	O	15000	7500	15	77	0	((0 0 -1500) 0.0)	75	0	-2	0
1-1-1-1-0	-150	O	15000	7500	15	3	0	((0 0 0) -0.2)	4	0	1	0
0-0-1-2-0	150	P	15000	-7500	15	589	0	((0 0 -3500) 0.2)	6174	0	5585	0
1-0-1-2-0	-150	P	15000	-7500	15	9	0	((0 0 0) 0.0)	9	0	0	0
0-1-1-2-0	150	O	15000	-7500	15	103	0	((0 0 0) 0.0)	92	0	-11	0
1-1-1-2-0	-150	O	15000	-7500	15	3	0	((0 0 0) 0.0)	3	0	0	0
0-0-1-0-1	150	P	15000	0	45	53	0	((1000 0 500) 0.0)	25494	0	25441	0
1-0-1-0-1	-150	P	15000	0	45	4	0	((-1500 0 0) 0.0)	5	0	1	0
0-1-1-0-1	150	O	15000	0	45	96	0	((0 0 500) 0.0)	75	0	-21	0
1-1-1-0-1	-150	O	15000	0	45	0	0	((0 0 0) 0.0)	0	0	0	0
0-0-1-1-1	150	P	15000	7500	45	25	0	((2000 0 500) -0.6)	34065	0	34040	0
1-0-1-1-1	-150	P	15000	7500	45	164	0	((0 0 0) 0.0)	123	0	-41	0
0-1-1-1-1	150	O	15000	7500	45	86	0	((0 0 0) 0.0)	94	0	6	0
1-1-1-1-1	-150	O	15000	7500	45	0	0	((0 0 0) 0.0)	0	0	0	0
0-0-1-2-1	150	P	15000	-7500	45	1581	0	((0 0 -500) 0.0)	3661	0	2080	0
1-0-1-2-1	-150	P	15000	-7500	45	0	0	((0 0 0) 0.0)	0	0	0	0
0-1-1-2-1	150	O	15000	-7500	45	143	0	((0 0 0) 0.0)	0	0	-143	0
1-1-1-2-1	-150	O	15000	-7500	45	0	0	((0 0 0) 0.0)	0	0	0	0
0-0-1-0-2	150	P	15000	0	75	18	0	((500 0 -1500) 0.4)	192119	0	192101	0
1-0-1-0-2	-150	P	15000	0	75	6	0	((-500 0 0) 0.0)	8	0	2	0
0-1-1-0-2	150	O	15000	0	75	68	0	((0 0 -7000) 0.0)	67	0	-1	0
1-1-1-0-2	-150	O	15000	0	75	0	0	((0 0 0) 0.0)	0	0	0	0
0-0-1-1-2	150	P	15000	7500	75	7	0	((-2000 0 0) -0.8)	7844	0	7837	0
1-0-1-1-2	-150	P	15000	7500	75	112	0	((0 0 -1000) 0.0)	103	0	-9	0
0-1-1-1-2	150	O	15000	7500	75	45	0	((0 0 -2500) 0.0)	58	0	13	0
1-1-1-1-2	-150	O	15000	7500	75	192	0	((500 0 -6500) 0.0)	155	0	-37	0
0-0-1-2-2	150	P	15000	-7500	75	94	0	((4500 0 -2000) 0.0)	521784	0	521690	0
1-0-1-2-2	-150	P	15000	-7500	75	0	0	((0 0 0) 0.0)	0	0	0	0
0-1-1-2-2	150	O	15000	-7500	75	110	0	((0 0 1000) 0.0)	0	0	-110	0
1-1-1-2-2	-150	O	15000	-7500	75	0	0	((0 0 0) 0.0)	0	0	0	0
0-0-1-0-3	150	P	15000	0	105	0	10	((0 0 0) 0.0)	14	0	14	-10
1-0-1-0-3	-150	P	15000	0	105	38	0	((0 0 -500) 0.2)	39	0	1	0
0-1-1-0-3	150	O	15000	0	105	19	0	((0 0 -1500) 0.0)	72	0	53	0
1-1-1-0-3	-150	O	15000	0	105	36	0	((0 0 500) 0.0)	35	0	-1	0
0-0-1-1-3	150	P	15000	7500	105	0	13	((0 0 500) 0.0)	8	0	8	-13
1-0-1-1-3	-150	P	15000	7500	105	79	0	((0 1000 -1000) 0.0)	62	0	-17	0
0-1-1-1-3	150	O	15000	7500	105	14	12	((0 0 0) 0.0)	46	0	32	-12
1-1-1-1-3	-150	O	15000	7500	105	90	0	((0 500 0) 0.0)	68	0	-22	0
0-0-1-2-3	150	P	15000	-7500	105	0	0	((0 0 -500) 0.0)	62	0	62	0
1-0-1-2-3	-150	P	15000	-7500	105	0	0	((0 0 0) 0.0)	0	0	0	0
0-1-1-2-3	150	O	15000	-7500	105	67	0	((0 0 -500) 0.2)	47	0	-20	0
1-1-1-2-3	-150	O	15000	-7500	105	0	0	((0 0 0) 0.0)	0	0	0	0
0-0-1-0-4	150	P	15000	0	135	0	9	((0 0 1000) 0.0)	14	0	14	-9
1-0-1-0-4	-150	P	15000	0	135	67	0	((-1000 6000 0) 0.0)	522	0	455	0
0-1-1-0-4	150	O	15000	0	135	0	17	((0 0 0) 0.0)	3	0	3	-17
1-1-1-0-4	-150	O	15000	0	135	66	0	((0 0 -1500) 1.0)	6095	0	6029	0
0-0-1-1-4	150	P	15000	7500	135	0	15	((-500 0 1000) 0.0)	16	0	16	15
1-0-1-1-4	-150	P	15000	7500	135	49	0	((0 0 -1000) 0.0)	91	0	42	0
0-1-1-1-4	150	O	15000	7500	135	0	14	((0 0 0) 0.0)	0	0	0	-14
1-1-1-1-4	-150	O	15000	7500	135	45	0	((0 0 -500) 0.2)	98	0	53	0
0-0-1-2-4	150	P	15000	-7500	135	0	14	((0 0 0) 0.0)	0	0	0	-14
1-0-1-2-4	-150	P	15000	-7500	135	4	0	((0 0 1000) -0.4)	7	0	3	0
0-1-1-2-4	150	O	15000	-7500	135	0	0	((0 0 1000) 0.0)	69	0	69	0
1-1-1-2-4	-150	O	15000	-7500	135	0	0	((0 0 0) 0.0)	0	0	0	0
0-0-1-0-5	150	P	15000	0	165	3	8	((0 0 -500) 0.0)	72	8	69	0
1-0-1-0-5	-150	P	15000	0	165	92	14	((0 0 -6500) 0.0)	155802	8	155710	-6
0-1-1-0-5	150	O	15000	0	165	14	11	((0 0 0) 0.0)	151	11	137	0
1-1-1-0-5	-150	O	15000	0	165	98	14	((0 0 -500) 0.0)	1000943	14	11000845	0
0-0-1-1-5	150	P	15000	7500	165	4	3	((0 0 -500) 0.0)	60814	3	60810	0
1-0-1-1-5	-150	P	15000	7500	165	0	0	((0 0 0) 0.0)	0	0	0	0
0-1-1-1-5	150	O	15000	7500	165	1	15	((0 0 1000) 0.0)	3	0	2	-15
1-1-1-1-5	-150	O	15000	7500	165	1	0	((0 0 0) 0.0)	1	0	0	0
0-0-1-2-5	150	P	15000	-7500	165	4	2	((0 0 0) 0.0)	28	2	24	0
1-0-1-2-5	-150	P	15000	-7500	165	30	0	((0 500 0) 0.2)	46	0	16	0
0-1-1-2-5	150	O	15000	-7500	165	2	3	((500 1000 -2000) -0.8)	35811	0	35809	-3
1-1-1-2-5	-150	O	15000	-7500	165	20	0	((-1500 -1500 500) 0.0)	26	0	6	0

Table 5.10 **Results of learning in distant range plan sectors.**

time when a standard MAXIM agent had nearly zero shot time in the same scenario. In the distant range plan sectors of Table 5.10 NOSTRUM was often able to decrease the agent's time in danger as it simultaneously increased shot time. In all but three distant range scenarios, NOSTRUM was able to reduce the time in danger to zero.

There were 20 training scenarios where learning actually had a negative impact on agent shot time, compared to what an agent using only proportional control would have in the same situation. During the first learning cycle within each plan sector NOSTRUM tested the default response to establish a baseline for performance. From this baseline, NOSTRUM was able to improve the shot time in some of these 20 sectors, but the net time after learning was still less than that available to a proportionally-controlled agent. For example, refer to the learning results from plan sector 1-0-0-1-3 in Table 5.9, shown in a bold-face font. Using strictly proportional control in each plan sector, the standard agent was able to accumulate 102 seconds of shot time. After learning, however, the total shot time for the flexible agent had dropped to 63 seconds. This drop in shot time can be attributed to responses learned in other plan sectors. These responses worked well when the scenario was initialized with the agent in position to use that response, but they sometimes had a negative impact when the agent entered from another sector. Since the total shot time increase in the 96 successful plan sectors far outweighed these marginal decreases I considered the losses to be negligible.

5.3 Testing the Features Hypothesis

In Section 1.3 a secondary hypothesis called the Features Hypothesis was presented to state my belief that agent behavior could be improved by selecting responses using a handful of key features. This hypothesis was expounded in subsequent sections to identify the five parameters that I selected as the key features and the resolution assigned to each, effectively dividing up the airspace around the target into a number of plan sectors. An investigation was not performed to determine which combat parameters should be selected as the key features; instead, I chose several features that I thought might be appropriate and conducted tests to assess the usefulness of these features. Section 5.2 discussed the effects of learning applied to each plan sector individually, but so far no mention has been

made to the overall changes in agent behavior. Examining how the flexible agent behaved in more realistic scenarios, rather than engagements against a non-jinking training drone, is the focus of this section.

Before an analysis of this type can be performed an important question must be answered: how shall "improvements" in behavior be measured? Improvements can be measured *qualitatively* by comparing tactics used by the agent to what might be employed by a human pilot. Since one of my goals was to make the agent behave more realistically we might say that behavior has improved if the tactics of a flexible agent after learning more closely resemble the actions of a human pilot. Alternatively, performance can be measured *quantitatively* by counting the number of kills accumulated by a flexible agent in a relatively large population of random scenarios against a proportionally controlled agent. Using this metric we might say that the agent's behavior has improved if it can destroy more enemy aircraft. I was curious to see what kind of improvements, if any, had resulted using each definition.

5.3.1 Qualitative Effect of Learning. To observe the qualitative effect of learning on overall agent behavior I created a battery of engagement scenarios. Each scenario was a 1-v-1 engagement pitting a standard MAXIM agent against either a flexible agent or another standard agent. Figure 5.10 shows the relative arrangement of the agent and target in each of the four scenarios. The initial conditions of each scenario were customized to allow for variations in agent to target separation, and to set the magnitude of the agent's velocity vector by some amount above or below the target's velocity. Each scenario was then run using the MAXIM simulator with learning disabled, alternating between standard and flexible agents as the opponent. Appendix B contains plots illustrating the behavior of each type of agent elicited during the scenarios. The behavior of the flexible agent in some of these scenarios is worth noting.

In the scenario depicted in Figure 5.11, the agent was initially behind and traveling more slowly than the target aircraft. The proportionally controlled aircraft immediately started to turn toward the enemy aircraft as soon as it was identified as a threat, shortly before time *t1*. On the other hand, the flexible agent recognized that it was moving more

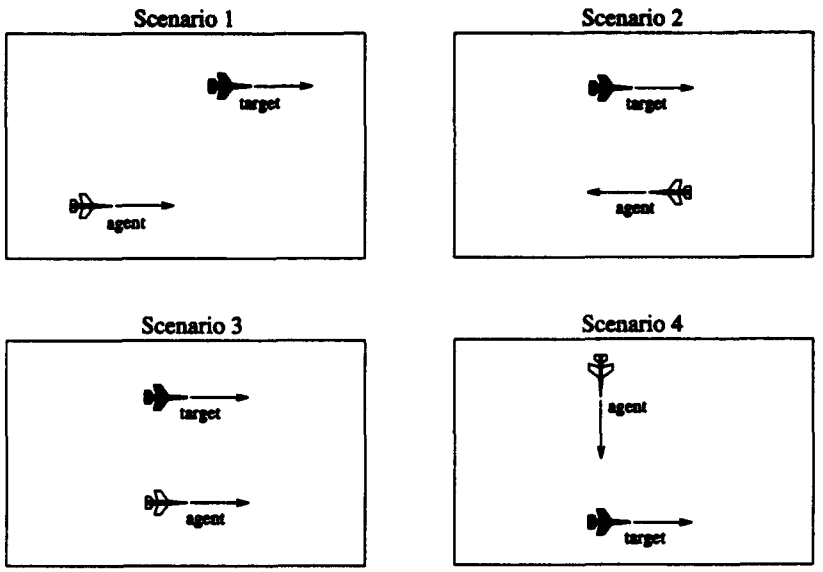

Figure 5.10 **Engagement scenarios.**

slowly than the target aircraft and tried to close in on the target using lead pursuit. Shaw says that lead pursuit is a tactic often used by pilots to increase closure on the target by use of geometry (28). Rather than fly towards the current location of a faster moving target, a pilot may instead fly towards the spot the target is *expected* to be a moment or two in the future. The flexible agent had been trained against a level-flying drone with this tactic in mind, and the tactic was transferred to the jinking drone in this scenario.

The behavior of the flexible agent in Figure 5.11 is an improvement over that of the standard agent for another, more subtle, reason. During combat, a pilot struggles to keep the target aircraft in view at all times because failure to do so can be fatal. Depending on how tightly the pilot would have to turn the aircraft, the maneuvers of the standard agent in this scenario might not be used by a human pilot because of the potential for losing sight of the target during the "blind" inside turn (28). If the agent aircraft was banking sharply during the turn, the aircraft would be dangerously exposing its underside while the line-of-site to the target was obscured. In contrast, the flexible agent made only outside

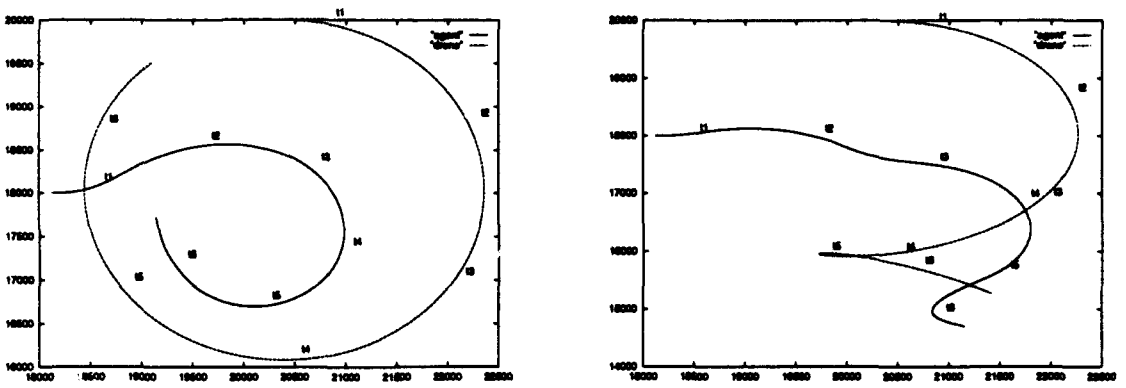

Figure 5.11 **Scenario #1, close range.** Two-dimensional behavior of standard (left plot) and flexible (right plot) agents.

turns and would probably have had the target continuously in view. This was considered an unexpected benefit, since the flexible agent was trained without any consideration given to this combat factor.

The initial conditions of Scenario #3 were similar to those of Scenario #1, except this time both agents started at the same X position in space. Figure 5.12 and 5.13 show the two- and three-dimensional plots of the resulting behavior for each agent type. Again, the standard agent flies towards the current location of the target and eventually crosses the target path sometime after time *t3*. The flexible agent not only tries to lead the target, but also dives slightly to gain airspeed as it tries to maneuver behind for a better position. The three-dimensional plot of the flexible agent in Figure 5.13 shows the agent and target jockeying for position in a downward scissors. In contrast, the three-dimensional plot of the standard agent in Figure 5.12 shows the competing standard agents dog-fighting almost entirely at a single altitude (note the Z axis scale in the plot).

The flexible agent also applied learned knowledge regarding climbs and deceleration to more dynamic situations. For example, in Scenario #2, shown in Figures 5.14 and 5.15, the agent initially started at close range with a greater velocity than the target aircraft. As in the previous scenario, the standard agent maneuvered very nearly within the same altitude throughout the simulation, but the flexible agent attempted to bleed off airspeed

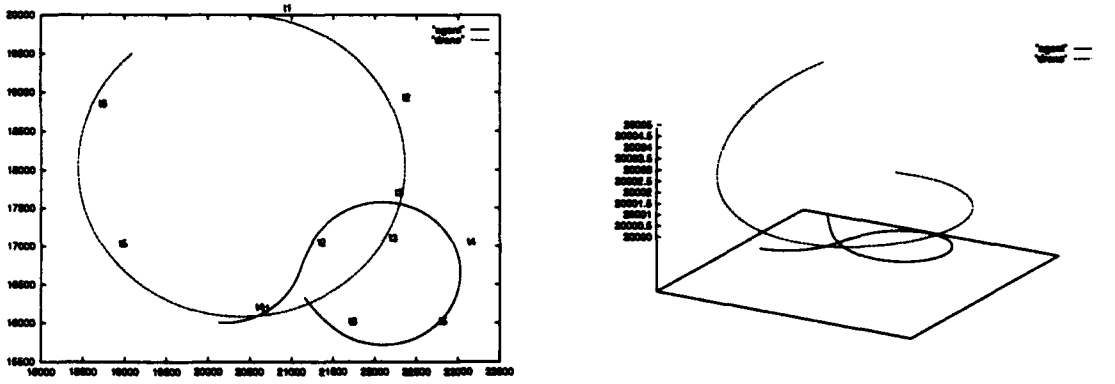

Figure 5.12 **Scenario #3, standard agent.** Two- and three-dimensional behavior of standard agent at close range.

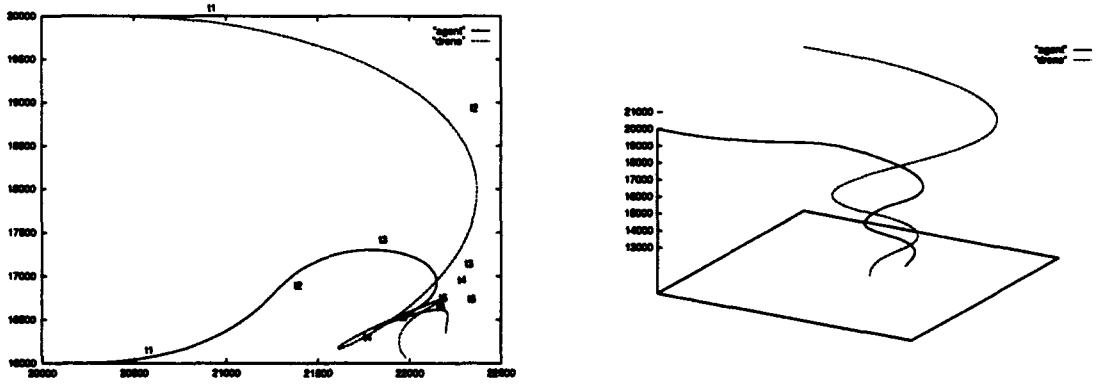

Figure 5.13 **Scenario #3, flexible agent.** Two- and three-dimensional behavior of flexible agent at close range.

through a climb 1000 m above the starting altitude. The flexible agent's climb, coupled with a negative power setting, resulted in a reduced overshoot to a maximum Y position of $\approx$ 21000 m, while the standard agent shot over 4000 m farther to a maximum Y position of $\approx$ 25000 m. Shaw speaks a good deal about maneuvers that trade airspeed for altitude, and the three-dimensional behavior of the flexible agent depicted in Figure 5.15 more closely resembles what a human pilot might do during a dog-fight.

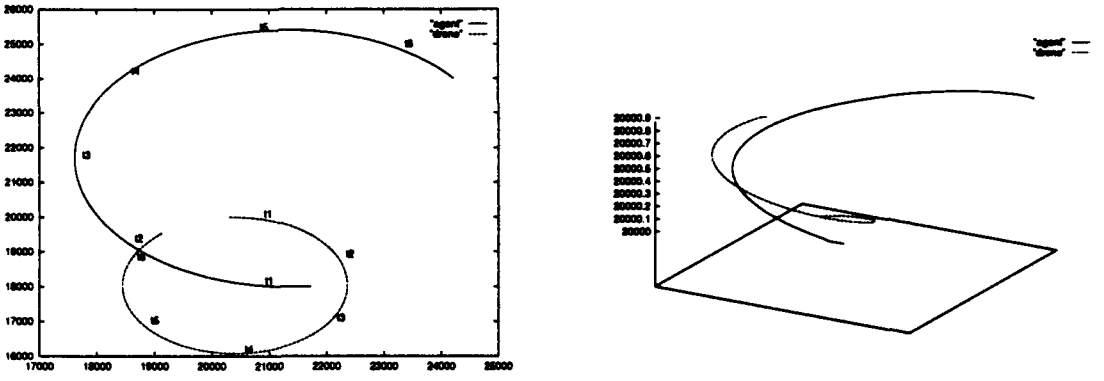

Figure 5.14 **Scenario #2, standard agent.** Two- and three-dimensional behavior of standard agent at close range.

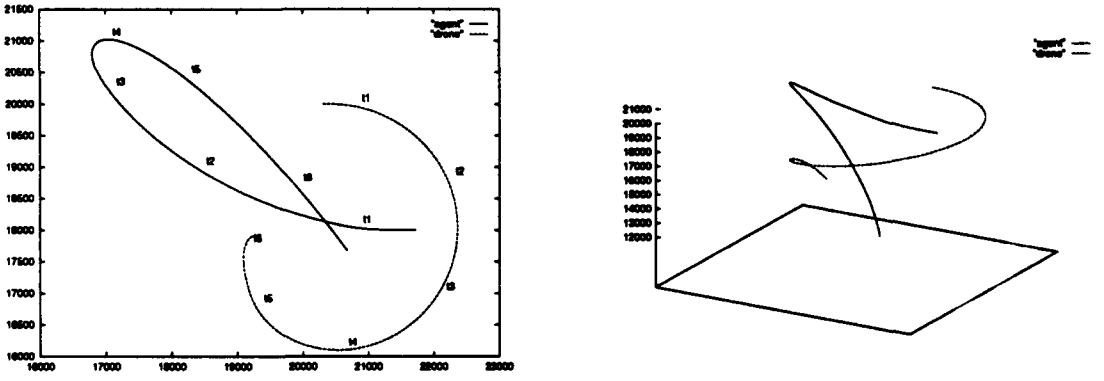

Figure 5.15 **Scenario #2, flexible agent.** Two- and three-dimensional behavior of flexible agent at close range.

An unanticipated *defensive* behavior emerged after learning offensive tactics. Figure 5.16 show the flexible agent engaged against the target in Scenario #1, but this time starting at long distance range with the agent moving slower than the target. The two aircraft started a collision course towards one another, but the flexible agent immediately began a spiraling nose dive shortly after flying past the target. Shaw refers to this maneuver as the *defensive spiral.* The defensive spiral is a very tight rolling scissors, normally initiated by a slower moving aircraft when an attacker is behind and in position for a shot. A pilot will use the defensive spiral to reverse the situation, forcing the faster moving attacker to overshoot during the dive and placing the defender in a better offensive position. Although the conditions prior to the defensive spiral did not warrant such a maneuver it was interesting to see this flexible agent behavior emerge as a side effect.

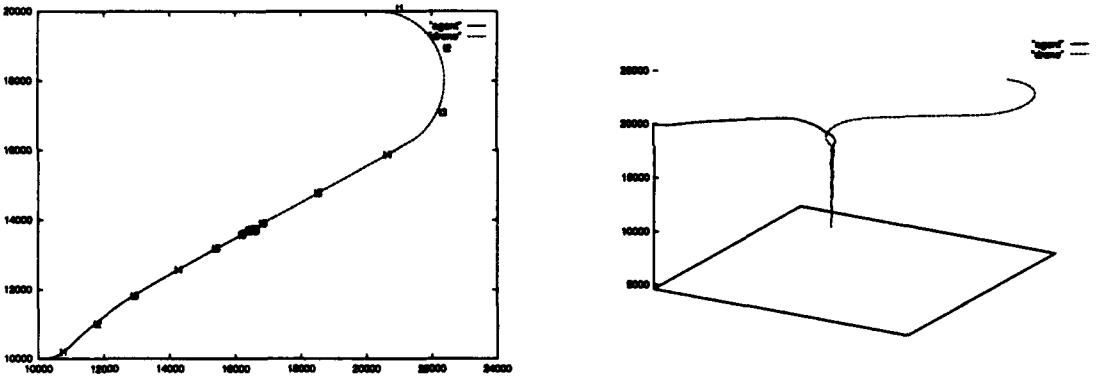

Figure 5.16 **Scenario #1, long-distance range.** Two-dimensional behavior of standard (left plot) and flexible (right plot) agents. The flexible agent demonstrated an unanticipated defensive maneuver known as the *defensive spiral.*

Ironically, Figure 5.16 also illustrates a learning shortcoming. I had hoped that NOSTRUM would discover better responses to the opposing heading-crossing angle plan sectors than the head-on collision tactic depicted in the plot. Rather than flying directly towards the opponent, NOSTRUM explored responses that moved the fly-to point by some amount to the left and right of the target aircraft position. This would then force the flexible agent to fly a course parallel to the oncoming target aircraft, putting it in a better position for an eventual turnaround and rear position shot. Unfortunately, NOSTRUM

never fully explored these responses because they always seemed less interesting than other responses.

As a final exercise, I wanted to see how the flexible agent behaved in the flat scissors scenario presented in Section 1.1. Using the attack responses learned against the non-jinking drone, the flexible agent maneuvered as shown in Figure 5.17. Compared to the behavior of the standard agent, the flexible agent's maneuvers more closely resemble what Shaw claims a human pilot would do in the same situation.

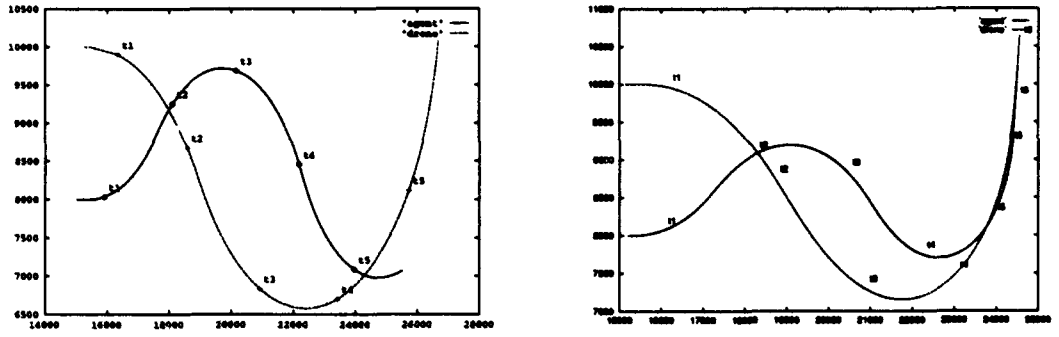

Figure 5.17 **Flat scissors, revisited.** Two-dimensional behavior of standard (left plot) and flexible (right plot) in the flat scissors scenario.

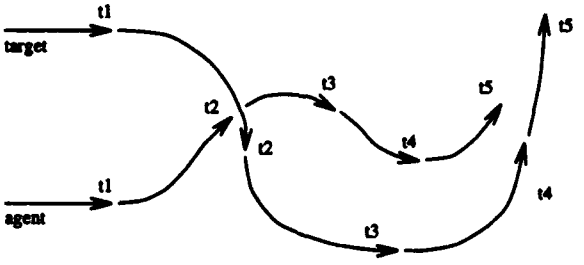

Figure 5.18 **Flat scissors, desired behavior.**

Determining the qualitative improvement in agent behavior based on knowledge learned in scenarios against a non-jinking drone was difficult. I developed the set of canned

scenarios to evaluate the flexible agent's success in applying the tactics to a dynamic environment, but this approach presumes many things. Each of the scenarios establishes an initial condition that may not be realistic and, consequently, may never occur in practice. The number of potential scenarios is infinite, and the subset used here to test behavior may not be a good representative sample. Furthermore, the scenarios were conducted for a finite length of time with both agents completely unarmed, making it difficult to identify the agent with the best end-game position. What the plots do indicate, however, is that many of the tactics learned by NOSTRUM against the non-jinking drone were successfully transferred to a dynamic environment.

The behavior of the flexible agent became much richer and less predictable than its standard agent counterpart. During several of these canned scenarios the two standard agents would circle each other endlessly when simulation time was extended, but this never happened when a flexible agent entered the game. Human pilots would not be expected to behave predictably, so the addition of response variety may also be considered a behavior improvement. It is important to recognize that response variations were not produced by random fluctuations in agent response, but was grounded in previous experience against a training drone. While we could always modify the standard agent to behave differently probabilistically, the flexible agent had a reason to respond the way it did[2].

5.3.2 Quantitative Effect of Learning. Determining the quantitative effect of learning was an easier task than attempting to determine qualitative improvements. This measurement was accomplished by running 500 randomly generated scenarios pitting a standard agent against a flexible agent with the learning advantage. Each scenario randomly selected agent and target starting positions, headings, and velocity magnitudes. Starting locations of the aircraft was restricted within a cubic arena 20000 m long on each side, and relative aircraft velocity was kept within ± 200 m/s. In addition, each aircraft was given four missiles to fire upon the other. In order to measure the effect of learning I ran the scenario first using a standard agent against the target. Then, the same scenario was run using a flexible agent.

[2]This is not to imply that the behavior of the flexible agent followed a chain of reasoning, but that it had tried different responses in somewhat similar situations and selected the best one it could find.

Results of 500 Random Scenarios		
Event	Agent Type	
	Standard Agent	Flexible Agent
Target Destroyed	191 times	186 times
Agent Destroyed	196 times	184 times

Table 5.11 **Results of 500 random scenarios.** Results of random scenarios used to measure quantitative improvement.

The results of the simulations, shown in Table 5.11, were not what I had expected. I had anticipated that the flexible agent, with the addition of responses employing various types of pursuits and energy management maneuvers, would out-kill the standard agent. Since the difference in the number of kills separating the standard and flexible agents is not statistically significant, the data indicates that learning may have had little quantitative impact on overall agent performance. However, the results of the 500 random scenarios does not reflect entirely upon the quality of the responses learned. NOSTRUM learned responses on the basis of improved shot and danger times in an environment completely free of enemy missiles. The underlying strategy was to improve the maneuvering capability of the agent so that it could maintain an offensive position, giving the agent multiple opportunities to fire upon and destroy the target if previously launched missiles were not successful. Once threatening missiles were introduced, however, the rules of the game were instantly changed. Immediately after the target aircraft launched a missile the agent transitioned out of *attack* mode and into *evade* mode. As a result, the random scenarios quite often tested the effectiveness of only a few seconds of the *attack* phase followed by many seconds in the *evade* phase. A better test of attack responses would restrict missiles to rear-quarter use. The flexible agent was trained to achieve a better rear-quarter position than the standard agent, yet the missile objects within MAXIM are all-aspect weapons. If missile use was limited to the rear quarter of a target aircraft, the flexible agent might be better suited to achieve a firing position than the standard agent. I did not pursue this method of

testing, however, because it entailed changing the rules of engagement to produce a desired result.

5.4 Summary

The discovery-based learning system, NOSTRUM, was designed to experiment with variations in attack responses for MAXIM air combat agents. I believed that DBL would be a useful tool for improving the behavior of an agent by training it to behave more appropriately in specific situations. Using improvements in shot and danger time as the discriminating factors, NOSTRUM was able to develop a more robust and less predictable set of universal plan responses for the flexible agent. Before learning, MAXIM agents pursued enemy aircraft using a single proportional control strategy. After learning, the flexible agent had learned the utility of lead, lag and pure pursuits, as well as how to use dives and climbs while maneuvering for a better offensive position.

I also believed that overall agent behavior could be improved by triggering responses using a handful of key features. This hypothesis was tested by engaging the flexible agent after learning against a standard agent. Qualitative and quantitative data was collected to determine whether or not the knowledge learned against a non-jinking drone would be transferable to a more dynamic environment, using the key features as indices into the array of responses. The flexible agent was trained in an environment devoid of weapons, so quantitative measurements of improved performance were not positive. However, many of the strategies learned during training were put into use while maneuvering against a jinking opponent. I consider this evidence to be supportive of a qualitative behavioral improvement.

VI. Summary, Conclusions & Recommendations

6.1 Research Summary

This thesis introduced NOSTRUM, a DBL system for exploring autonomous agent responses in the air combat domain. While there are many functional areas within air combat that could have been investigated, NOSTRUM focused its attention to tactics that might be useful during a dog-fight with an enemy aircraft. NOSTRUM was tuned for experimentation in the world created by MAXIM, a prototypical air combat simulator developed at AFIT for potential use in the DIS project.

Several autonomous agent architectures were reviewed, with attention given to reactive, planning, deliberative, and combined systems. MAXIM is a reactive system based on the concept of the universal plan. A universal plan is simply a set of pre-planned agent responses designed to cover every possible situation the agent might encounter. The universal plan employed by MAXIM is limited in its variety of responses, sometimes resulting in predictable or arbitrarily inappropriate behavior.

NOSTRUM used discovery-based learning to enhance one portion of MAXIM's universal plan, the part dictating behavior during the *attack* phase. A new representation of the universal plan was created and integrated into MAXIM, partitioning behavior into 144 plan sectors. Instead of relying exclusively on the proportional control strategy previously hard-coded into the MAXIM universal plan, more appropriate responses were chosen based on the values of the key features. Five parameters were selected as the key engagement features: aspect angle, relative altitude, relative velocity, heading crossing angle, and range to target. Each feature was used as an index into a table of attack responses. This approach lead to the flexible agent, a less predictable autonomous agent designed to select the most appropriate response based on the combat situation of the moment.

Some machine learning techniques were examined, but discovery-based learning appeared to be the most appealing. The complexity of the air combat tactics domain, combined with my lack of expert knowledge in the domain, readily lent itself to a form of learning that was not inherently knowledge intensive. Compared to explanation-based learning, DBL systems typically require only a small amount of domain knowledge for

learning to begin. Instead, the discovery-based system acquires knowledge by mimicking the scientific method: the system examines data, formulates hypotheses, selects and conducts experiments to test hypotheses, and repeats.

The act of discovery used by NOSTRUM was a modified form of the scientific method. The system examined data reflecting the usefulness of a response and suggested modifications that might improve behavior. NOSTRUM then executed a scenario to evaluate a suggested response, and the cycle would repeat. As long as the behavior of the agent appeared to be improving NOSTRUM continued suggesting new responses to test. After a number of un-interesting experiments had been conducted, NOSTRUM sensed that progress was no longer being made towards an improved solution. At that point, it remembered the best response discovered for that plan sector and moved on to the next. In this fashion, NOSTRUM experimented with alternative attack responses for all 144 plan sectors.

Attack responses were represented as three-dimensional offsets to the current position of the target aircraft. An additional parameter in the response was used to adjust the aircraft power setting. During simulation execution, a flexible agent would retrieve the appropriate attack response from an array containing all 144 responses. The fly-to point was then "tailored" to the current position and heading of the target aircraft by adding offsets to the target nose, wing, and tail vector. NOSTRUM suggested variations in attack responses by adjusting the fly-to point in space relative to the target aircraft while simultaneously suggesting changes in the aircraft power setting. This contrasted with the standard agent attack response which always maneuvered the agent directly towards the enemy aircraft, giving no consideration to target bearing or the potential utility of energy management maneuvers.

6.2 Conclusions

This research was designed to test two sub-hypotheses I had formulated regarding the suitability of DBL as a tool for learning tactics in the air combat domain. My conclusions regarding each sub-hypothesis will be presented individually, starting with the DBL Hypothesis:

DBL Hypothesis:
Given the division of agent responses by key features, discovery-based learning can improve behavior by testing various responses within each division and remembering those that worked best.

The DBL Hypothesis was the foundation of this research. With it, I stated my belief that discovery-based learning could be used to improve the behavior of an autonomous agent, sector by sector. To test this hypothesis, NOSTRUM was programmed to cycle through all 144 plan sectors against a non-jinking drone, suggesting variations in each plan sector response and testing those that appeared to be the most promising. Sets of interestingness and response heuristics were developed to guide NOSTRUM through the search space as it looked for and remembered responses that improved shot and danger time.

The data presented in Chapter 5 indicate that discovery-based learning was useful in learning more appropriate responses to the training scenarios. In the overwhelming majority of the plan sectors, NOSTRUM was able to increase the available shot times through a combination of fly-to point and power setting adjustments. In a significant number of the training scenarios, NOSTRUM was able to place the flexible agent in a stable firing position behind the target for indefinitely long periods of time, whereas the standard agent shot time was definite and short. I consider this research objective successfully achieved.

A secondary issue was addressed along with the DBL Hypothesis:

- What domain knowledge is required for discovering air combat tactics?

I believe that the interestingness and response heuristics within NOSTRUM reflect some of the important domain knowledge for air combat tactics, but developing the heuristic sets to be general enough for the many training scenarios was difficult. Throughout system development and testing, I had to resist the temptation to add heuristics that would guide the system along a desired path. This was especially true for the plan sectors with opposing heading crossing angles and large aspect angles, referred to previously as the

"dirty" sectors. While it is virtually impossible to totally eliminate bias in the heuristics of a DBL system, the addition of specialized heuristics would certainly diminish the successes of the system to discover better responses for itself.

Initially, I thought that my near total lack of combat tactics knowledge was going to be an impediment to in this research. As work progressed, however, I began to suspect that the gaps in my knowledge were almost beneficial. Critics of DBL systems contend that the heuristics are so carefully constructed that the program cannot help but stumble upon the desired solution. The question then arises: Did the system actually learn anything, or was something already locked into the heuristics *implicitly* just made explicit. Since I was not an expert in the domain it was very difficult for me to coach the system into producing any particular result. Although my heuristics encapsulate some very basic notions about energy management, as well as what the "good" and "bad" aspects of an engagement might be, the responses discovered by NOSTRUM were the result of an honest discovery process.

The likelihood of ever developing a minimal set of heuristics to deal with the wealth of possible combat scenarios, without coaching the system, seems remote. NOSTRUM failed to learn some responses as I had expected because it did not find the search path leading to these responses interesting. The best example is when NOSTRUM explored responses for the opposing heading-crossing angle sectors. In many of the training scenarios for these plan sectors, a human pilot might direct his aircraft along a flight path parallel to the target's flight path. Then, after the two aircraft have passed each other, the pilot would immediately turn towards the target six o'clock position. Maneuvering in this manner has qualitative advantages that are difficult to state in the form of an interestingness heuristic, nor is it easy to state unequivocally which response is "better" or "best." The payoff resulting from a particular maneuver is not always obvious, and sometimes the situation must temporarily degrade before the response that best resembles a human response can be found. In the situation presented here, NOSTRUM did not investigate fly-to point adjustments that made the flight paths more parallel because it did not increase, and sometimes reduced, shot time. Without explicitly checking for this situation and assigning an associated interestingness bonus NOSTRUM would never discover the textbook response.

The other sub-hypothesis investigated during this research was the Features Hypothesis:

Features Hypothesis:
Agent behavior can be improved by selecting a response based on a handful of key features.

This hypothesis stated my belief that overall agent behavior could be improved by selecting the most appropriate response based on a manageable set of key features. If the responses known by the agent were appropriate, then these key features could be used as indices into a universal plan. I tested this hypothesis using two techniques. The first method was used to gauge qualitative improvements in agent behavior by engaging a flexible agent against a jinking target aircraft in a battery of test scenarios. The second method used quantitative measurements to count the number of successful target kills by the flexible agent compared to the standard agent in a large population of randomly generated scenarios.

Chapter 5 and Appendix B present plots of the flexible agent behavior compared to the standard agent behavior in each of the 32 test scenarios. These plots indicate that some of the knowledge learned by the flexible agent against the non-jinking drone was transferable to a more dynamic environment populated with maneuvering enemy aircraft. I clearly saw cases where the flexible agent used variations in tactics against the jinking target, such as lead and lag pursuit and energy management strategies, that NOSTRUM had found to be effective against the non-jinking drone. The resultant behavior of the flexible agent was also sufficiently varied that predictability was effectively reduced. Consequently, I believe that the Features Hypothesis was also successfully confirmed.

Quantitative improvements in agent behavior were not noticeable. This was measured using 500 randomly generated scenarios pitting the flexible agent against a standard MAXIM agent. Each of the 500 scenarios was also run with two standard agents competing against each other to establish a performance baseline. A comparison of the number of target aircraft destroyed by the standard agent and those by the flexible agent indicate that

there was no statistical improvement in performance once the two aircraft were armed and allowed to fire upon one another. This measurement should not be used as an indication of the success or failure of DBL applied to this domain since the flexible agent was trained in an weapon-less environment in a single phase of execution whereas the scenarios exercised the agent in all three phases.

Associated with the Features Hypothesis were three related side issues:

- During combat, what parameters are critical when selecting a course of action?
- What granularity in the critical parameters is required to improve agent behavior?
- How should the behavior of MAXIM agents be represented so that a working learning component can be integrated into the simulator?

These three issues are all tightly intertwined in the representation I choose for the universal plan. The universal plan was decomposed into 144 smaller slices known as plan sectors. Each plan sector identified a response that was tuned to a specific engagement situation, and was selected using the five parameters as an index into a table of responses. When the flexible agent started in a position to the rear of the target aircraft this representation of the universal plan appeared to work well, accounting for most of the improvement in overall agent behavior in the test scenarios. However, when the flexible agent moved outside of the rear plan sectors and into dirtier sectors the representation was not as effective. Responses in opposing heading-crossing angle sectors were not executed for a significant period of time because the agent quickly moved from sector to sector. Likewise, scenarios that placed the agent in a starting position with a large aspect angle quickly degenerated into an opposing heading-crossing angle scenario as the agent turned backwards to meet the opponent.

The overall objective of this research was to test my primary hypothesis, that DBL could be used to improve the performance of an agent in the air combat domain. The primary hypothesis was decomposed into the aforementioned sub-hypotheses, and would be supported if each of the sub-hypotheses could be independently verified. I believe the

primary objective has also been accomplished, and that DBL was shown to be a useful tool for acquiring knowledge about tactics in the air combat domain.

6.3 Recommendations for Future Research

In the course of this work a few alternatives for future enhancement have been considered that I was not able to investigate. Some of these ideas relate to the discovery process in general, others for improving performance in the air combat domain in particular.

- DBL systems tend to be hill climbing systems, and NOSTRUM is no exception. A potential problem with this behavior is that the system can climb blindly up the first hill it sees and get stuck, when a much better solution could have been found if the system looked in every direction before taking a single step. Rather than being a pure hill climber, NOSTRUM could employ *steepest ascent hill climbing*, examining all spawned response and assess the interestingness of each. This would really force the system to progress along the path with the greatest prospect for success, instead of investigating the *first* path encountered that seemed interesting.
- The plan sector division of universal plan responses works well in a limited number of situations. The difficulty is actually the result of a frequently shifting fly-to point; in many situations, the flexible agent did not have enough time to execute the plan sector response for it to have an effect. One possible solution to this problem might be to continue selecting responses based on the current situation (as indicated by the key parameters), but execute the response for a fixed period of time. The DBL system could experiment with not only the responses but with the amount of time to continue applying the response. When the elapsed time has ended, the agent could re-assess the situation and select a new appropriate response to execute for the next few moments.
- Another approach that may improve the behavior of the agent would be to partition the space around the target into unequal slices, dividing the plan sectors more coarsely so that the dirty plan sectors are bigger than the pure plan sectors. With

a larger slice of airspace the flexible agent will remain in these sectors for a greater period of time, giving the response more time to have an impact.

- It may not be possible to produce behavior indistinguishable from humans in this domain without a much larger base of air combat knowledge to work from. DBL could still be leveraged, however, by using a more sophisticated approach: the system could have knowledge of many basic maneuvers and how to implement them, and discovery could be used to splice them together in an effective sequence. This would be labor intensive on the part of the developer, and would require considerably more expert knowledge to build the maneuver knowledge base.

6.4 Closing Thoughts

As NOSTRUM discovered improvements in its attack responses I was discovering how difficult it is to learn tactics that can be applied in dynamic situations. Getting NOSTRUM to learn responses that improved the behavior of the flexible agent in a single training scenario was relatively easy, but trying to develop heuristics that were general enough for every training scenario was another matter. Nevertheless, the exposure to the process of discovery was enlightening and useful.

I have reservations about the usefulness of DBL applied to anything but the most restricted domains. The supposed benefit of DBL is that only a small base of domain knowledge is required for learning to begin, but dynamic and unpredictable situations cannot be effectively handled without adding heuristics to specifically deal with these situations. This leads to system coaching, effectively forcing the DBL system to "discover" treasure when you already know where it is already buried. If this kind of coaching is required, of what use is discovery? In these situations, it might be simpler to build the desired knowledge immediately into the system and eliminate the time-consuming learning process [1].

[1] Although NOSTRUM was not built with speed and memory efficiency in mind, it was excruciatingly slow. A typical learning session took six days from start to finish. This was primarily due to execution time that accumulated while running thousands of simulations, each taking anywhere from 45 seconds to 90 seconds to complete.

Another important thing to consider is the desired comparability of agent behavior after learning to that of a human expert. It is quite possible for a DBL system such as NOSTRUM to improve behavior in the air combat domain, but the resultant behavior may not closely match the tactics actually employed by fighter pilots. Such systems may discover approaches to combat that do not have corresponding equivalents in military doctrine. If the goal is to build a tactical knowledge base for use in an interactive simulation it would be desirable for the final behavior to model that of the intended ally or enemy.

It was a challenge to implement discovery-based learning in a dynamic environment. In the air combat domain, it was difficult to isolate the parameters to collect and compare, and it was equally difficult to measure progress when it did not immediately result in improvements in shot or danger time. Discovery becomes more complicated when the world is populated by other agents, adding credit assignment to the list of potential problems. I made an effort to circumvent this difficulty by training the flexible agent against a predictable non-jinking drone. Agent behavior did improve after learning, but the responses are certainly brittle in the sense that they are tuned for very specific situations. This may be an indication that DBL is not suited for learning in environments that are ill-behaved and unpredictable.

Appendix A. Terms and Computations

This Appendix provides the details of the equations used by NOSTRUM to compute the values of the five key parameters: aspect angle, relative altitude, range, relative velocity, and heading-crossing angle. Details of fly-to point adjustments are also provided. Figure A.1 graphically depicts the terms involved in these calculations.

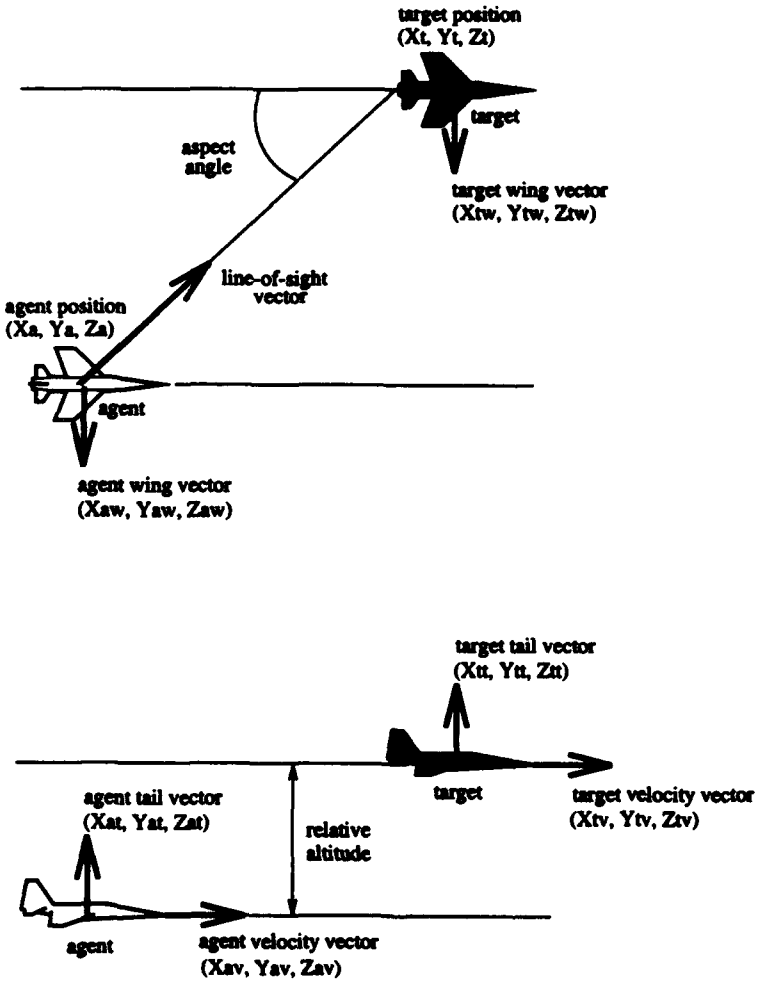

Figure A.1 **Terms used by NOSTRUM.**

The following terms are used in the equations to follow:

$$\overline{position_{agent}} = \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix}, \quad \overline{position_{target}} = \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}$$

$$\overline{velocity_{agent}} = \begin{bmatrix} x_{av} \\ y_{av} \\ z_{av} \end{bmatrix}, \quad \overline{velocity_{target}} = \begin{bmatrix} x_{tv} \\ y_{tv} \\ z_{tv} \end{bmatrix}$$

$$\overline{wing_{agent}} = \begin{bmatrix} x_{aw} \\ y_{aw} \\ z_{aw} \end{bmatrix}, \quad \overline{wing_{target}} = \begin{bmatrix} x_{tw} \\ y_{tw} \\ z_{tw} \end{bmatrix}$$

$$\overline{tail_{agent}} = \begin{bmatrix} x_{at} \\ y_{at} \\ z_{at} \end{bmatrix}, \quad \overline{tail_{target}} = \begin{bmatrix} x_{tt} \\ y_{tt} \\ z_{tt} \end{bmatrix}$$

A.1 Aspect angle

The aspect angle was used by NOSTRUM to partially describe the position of the agent relative to the target aircraft. This measurement does not account for the heading of the agent relative to the target, but does relate the position of the agent relative to the target heading. Aspect angle was computed by projecting the line-of-sight vector onto the plane passing through the target aircraft fuselage and wings, and measuring the angle between this projection and an imaginary line extending from the rear of the target, as shown in Figure A.1.

NOSTRUM computes the aspect angle as follows:

$$\overline{los} = \overline{position_{target}} - \overline{position_{agent}}$$

$$los_{wing} = \overline{los} \cdot \overline{wing_{target}}$$

$$los_{nose} = \overline{los} \cdot \overline{velocity_{target}}$$

$$\tan(aspect) = \frac{los_{wing}}{los_{nose}}$$

Valid values for aspect angle range from 0 to 359 degrees. The $\tan(aspect)$ computation resolves the aspect angle to a value between 0 and 180 degrees; NOSTRUM does additional computation to resolve the angle into the proper quadrant, as shown in Figure A.2.

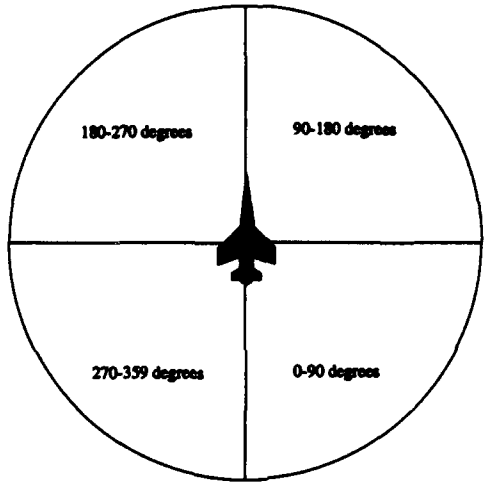

Figure A.2 Aspect angle quadrants.

A.2 Relative Altitude

Relative altitude was used to describe the relative position of the agent above or below the target aircraft. This value was the simplest feature value to compute:

$$altitude_{relative} = z_a - z_t$$

A positive $altitude_{relative}$ indicated that the agent was above the target.

A.3 Heading-Crossing Angle

The heading-crossing angle, *hca*, was computed using the vector dot product. An *hca* of 0 degrees indicated that the agent was flying a course parallel to that of the target; an *hca* of 180 degrees indicated that the agent was flying a course in the opposite direction.

$$\cos(hca) = \frac{\overline{velocity_{agent}} \cdot \overline{velocity_{target}}}{\|\overline{velocity_{agent}}\| \, \|\overline{velocity_{target}}\|}$$

A.4 Range

Range, the distance between the target and agent, was calculated by determining the magnitude of the agent line-of-site vector:

$$range = \sqrt{(x_t - x_a)^2 + (y_t - y_a)^2 + (z_t - z_a)^2}$$

A.5 Relative Velocity

Relative velocity was used to indicate how much faster or slower than the target aircraft the agent was moving. A positive $velocity_{relative}$ indicated that the agent is moving faster than the target.

$$velocity_{relative} = \|\overline{velocity_{agent}}\| - \|\overline{velocity_{target}}\|$$

A.6 Fly-to Point Adjustments

This section provides the details of fly-to point adjustment, the process of rotating three-dimensional response offsets onto the world coordinate system. To begin, NOSTRUM required an additional piece of information not accounted for in Figure A.1: the local coordinate system response.

$$response \equiv ((offset_{nose} \quad offset_{wing} \quad offset_{tail}) \quad power)$$

The nose and wing offsets in the response, $offset_{nose}$ and $offset_{wing}$ specified a component of the right-handed coordinate system local to the target aircraft, as shown in

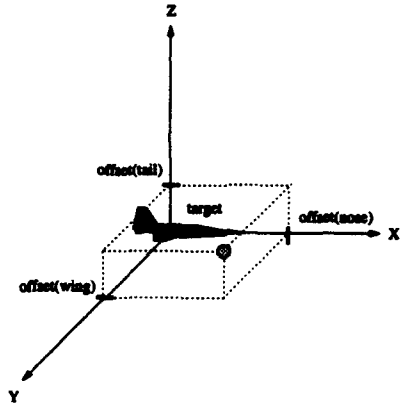

Figure A.3 **The local target coordinate system.**

Figure A.3. The tail offset, $offset_{tail}$, specified a displacement above or below the target position on an axis that was parallel to the world coordinate system Z axis. The $offset_{tail}$ term was handled differently because I wanted a flexible agent to climb or dive relative to the target position, regardless of target orientation; if this offset was allowed to rotate with the orientation of the target, a response that would normally warrant a climb would become a dive for inverted targets.

To prevent intended climbs and dives relative to the target from having the opposite effect, NOSTRUM calculates a modified heading vector that takes the X and Y components of the target velocity vector into account:

$$\overline{heading_{target(modified)}} = \frac{\begin{bmatrix} x_{tv} \\ y_{tv} \\ 0 \end{bmatrix}}{\|\overline{velocity_{target}}\|} = \begin{bmatrix} x_h \\ y_h \\ 0 \end{bmatrix}$$

NOSTRUM uses the modified heading vector to compute a modified wing vector. This was done by crossing the modified heading vector into the Z axis to produce a right-handed orthogonal vector component:

$$\overrightarrow{wing_{target(modified)}} = \overrightarrow{heading_{target(modified)}} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix}$$

Calculating the position of the fly-to point in the world coordinate air space was accomplished by adding the offsets in the local airspace, individually rotated, to the current target position:

$$\overrightarrow{flyToPoint_{rotated}} = \overrightarrow{position_{target}} + offset_{nose}\overrightarrow{heading_{target(modified)}} + offset_{wing}\overrightarrow{wing_{target(modified)}} + offset_{tail}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Appendix B. Engagement Scenarios

This appendix illustrates the behavior of the standard and flexible agents in each of the four scenarios presented in Chapter 5. Each scenario was executed four times using the following starting parameters:

1. Close range, agent moving slower than target
2. Close range, agent moving faster than target
3. Long-distance range, agent moving slower than target
4. Long-distance range, agent moving faster than target

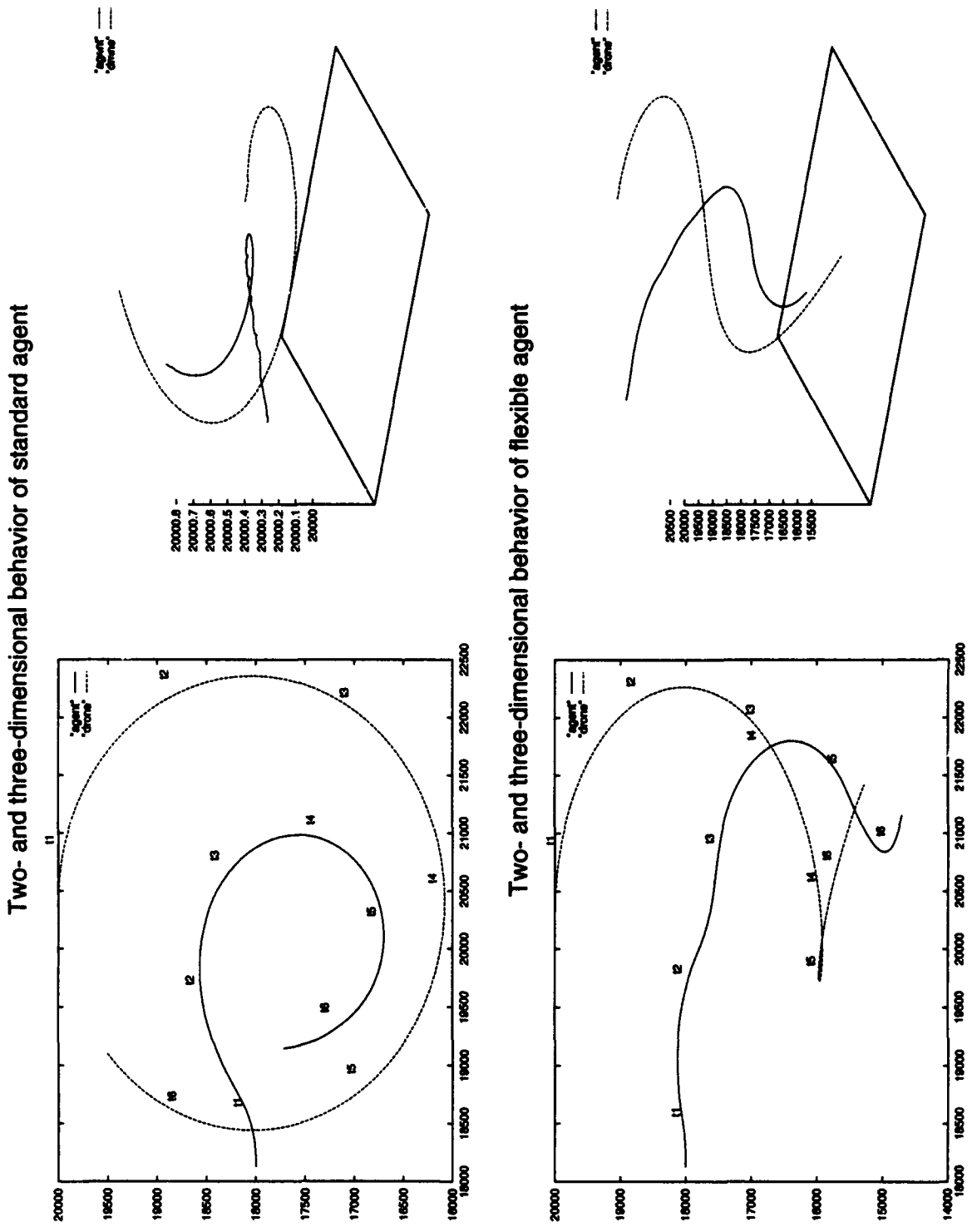

Figure B.1 **Scenario #1, close range, agent moving slower than target.**

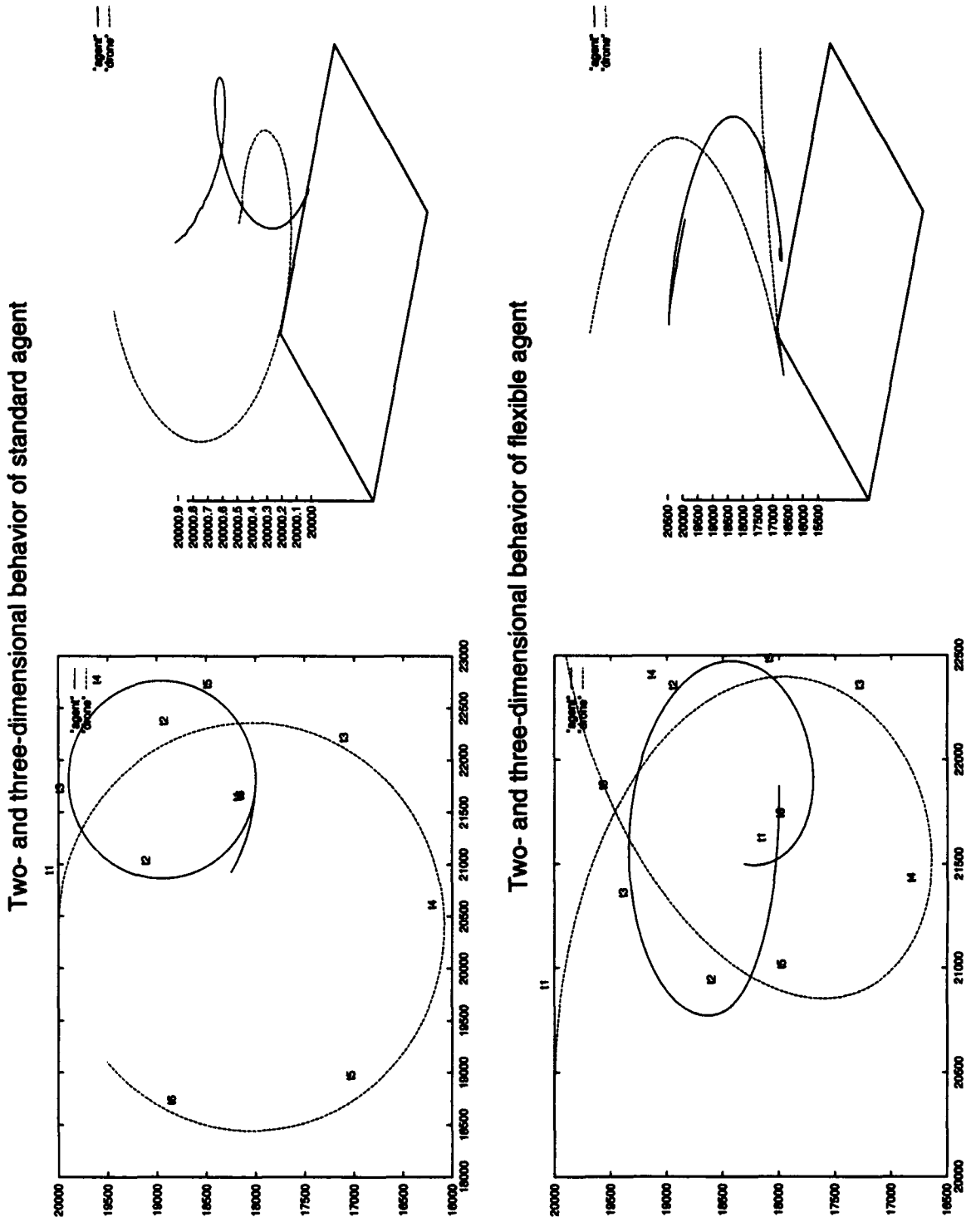

Figure B.2 **Scenario #2, close range, agent moving slower than target.**

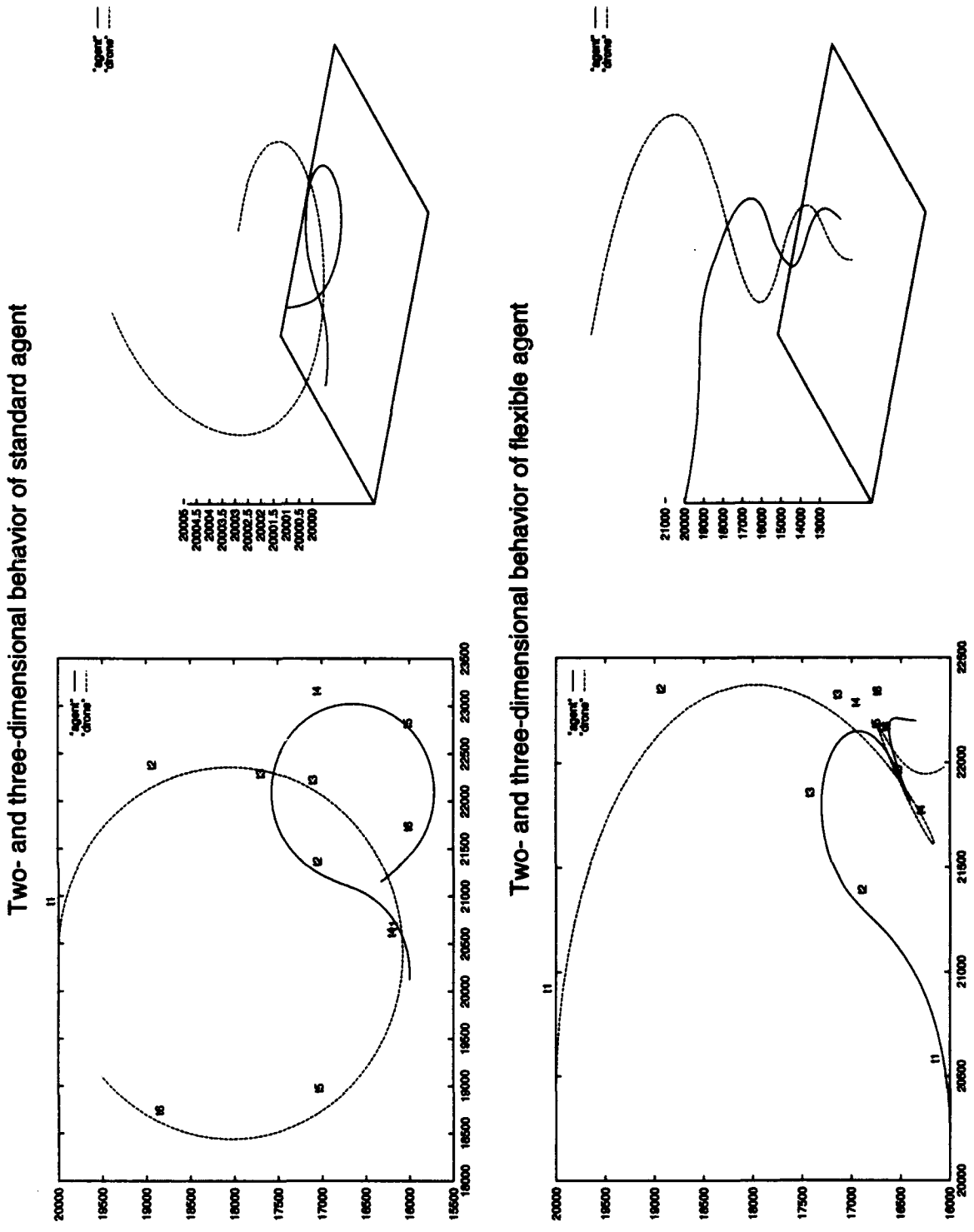

Figure B.3 **Scenario #3, close range, agent moving slower than target.**

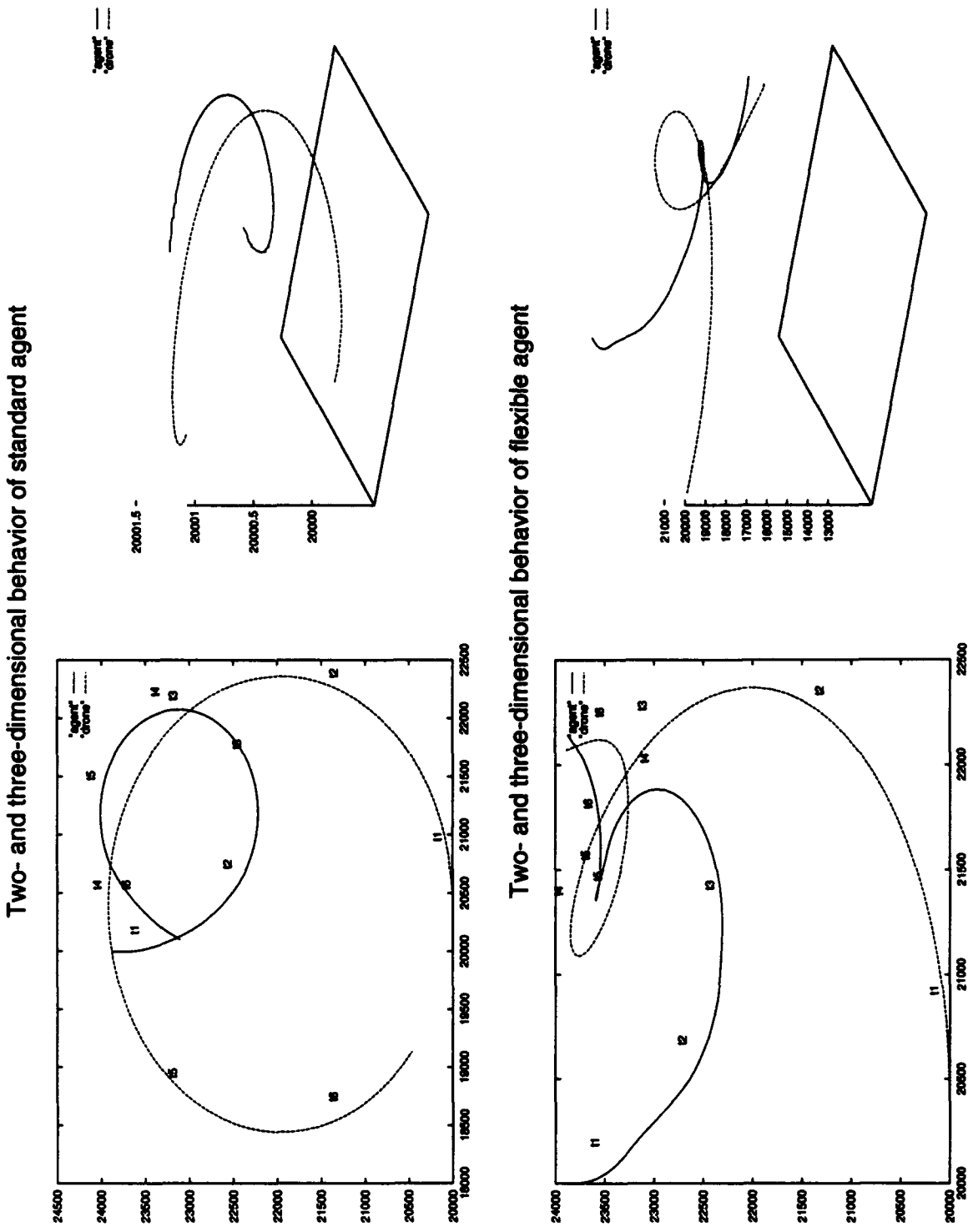

Figure B.4 **Scenario #4, close range, agent moving slower than target.**

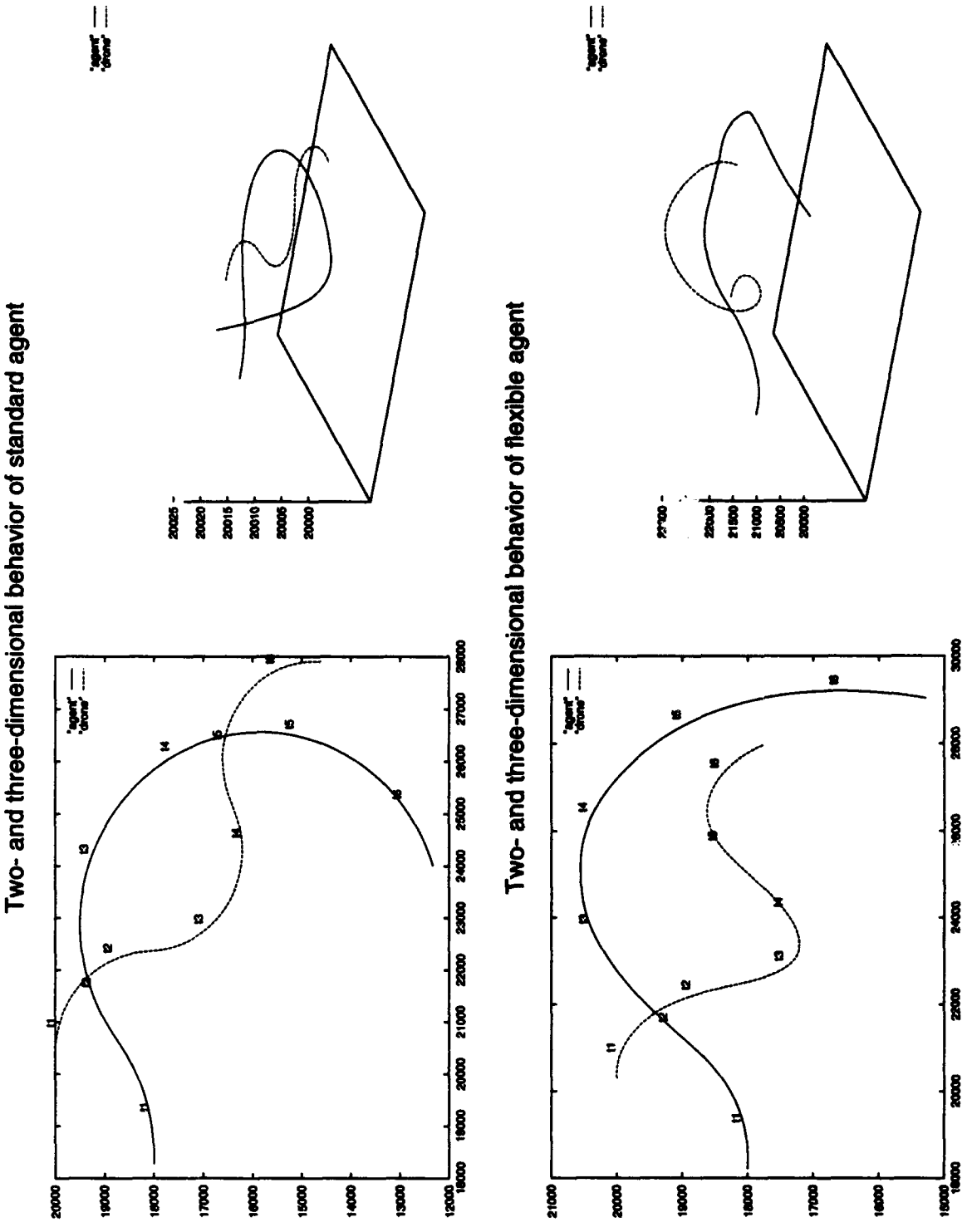

Figure B.5 **Scenario #1, close range, agent moving faster than target.**

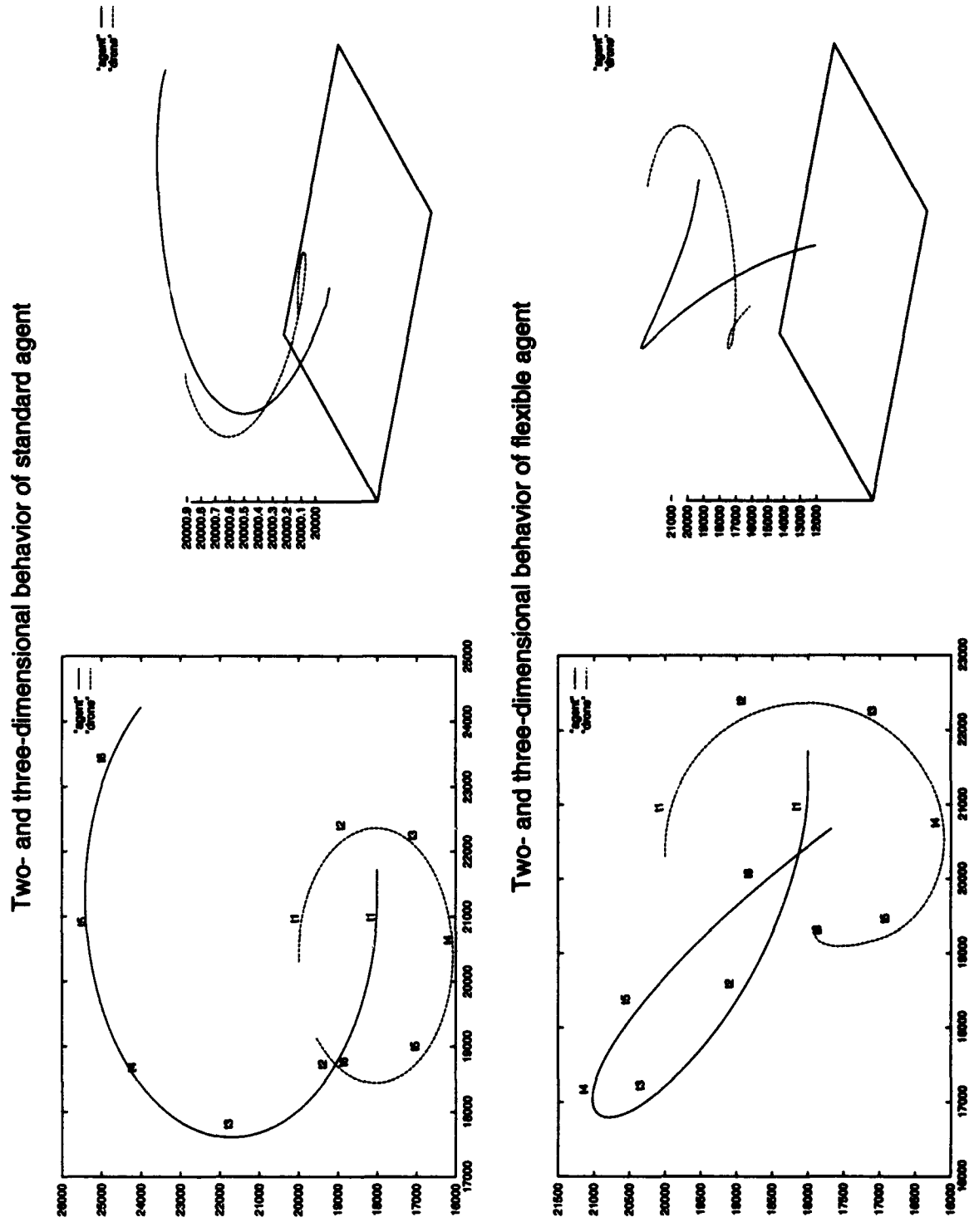

Figure B.6 **Scenario #2, close range, agent moving faster than target.**

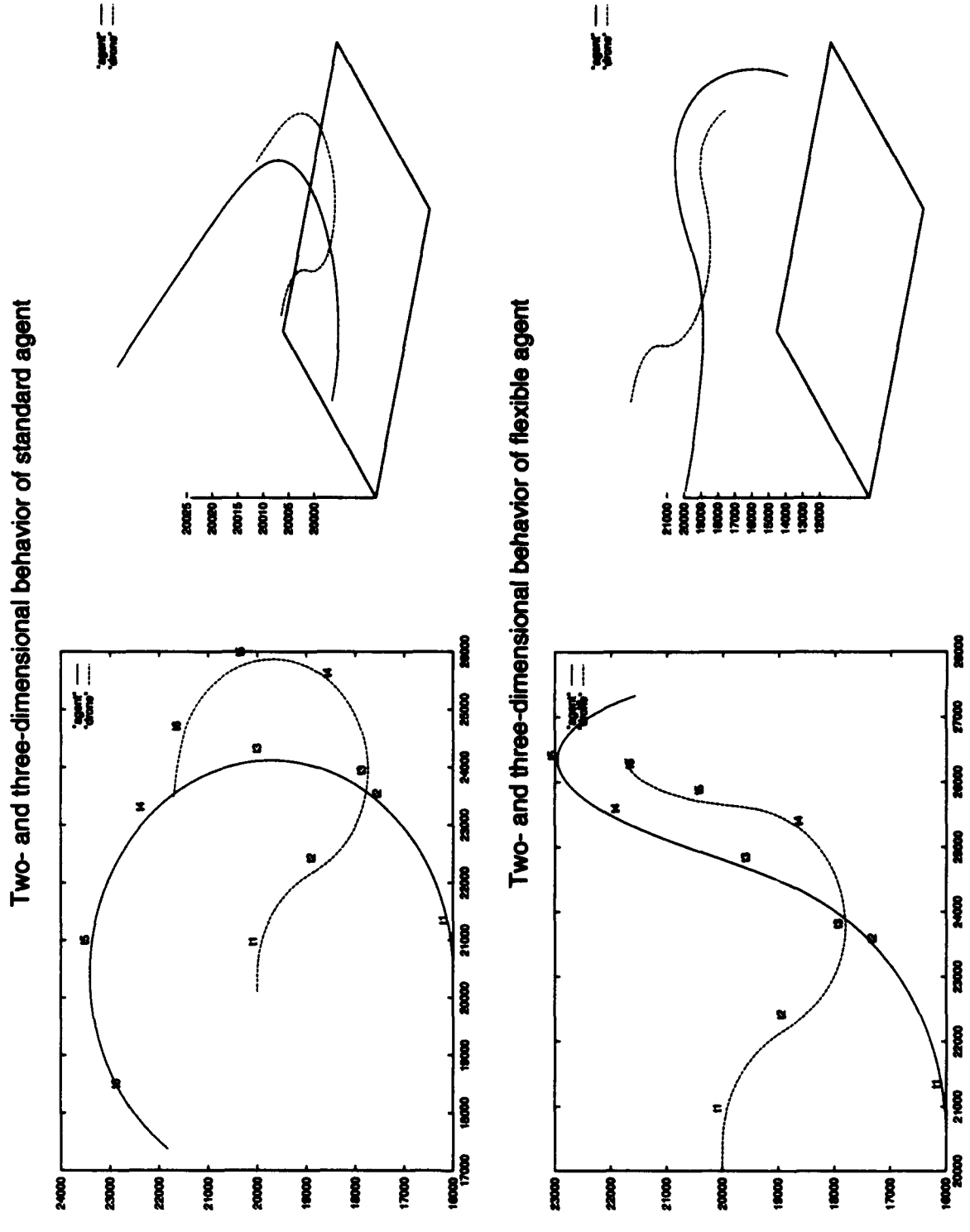

Figure B.7 **Scenario #3, close range, agent moving faster than target.**

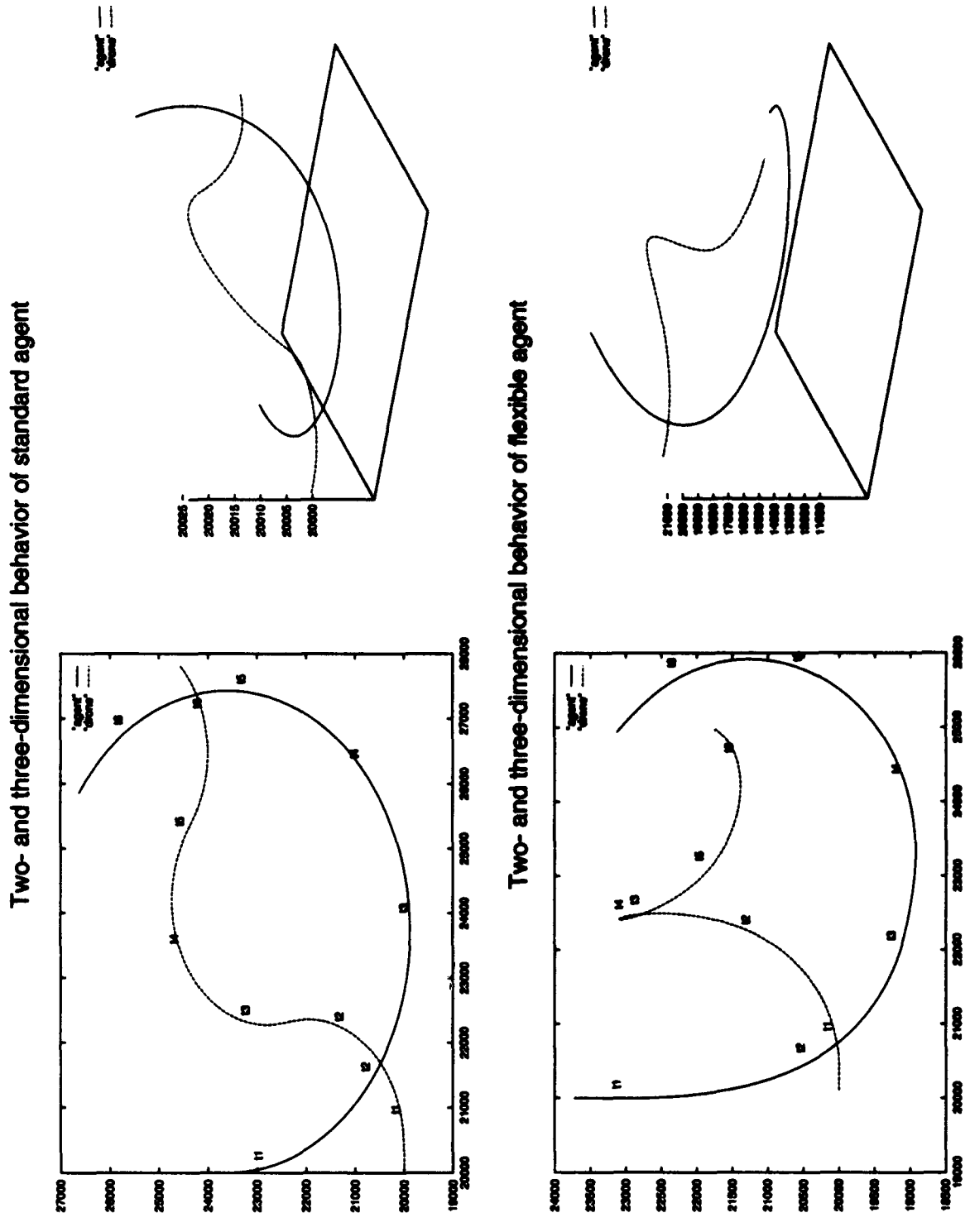

Figure B.8 **Scenario #4, close range, agent moving faster than target.**

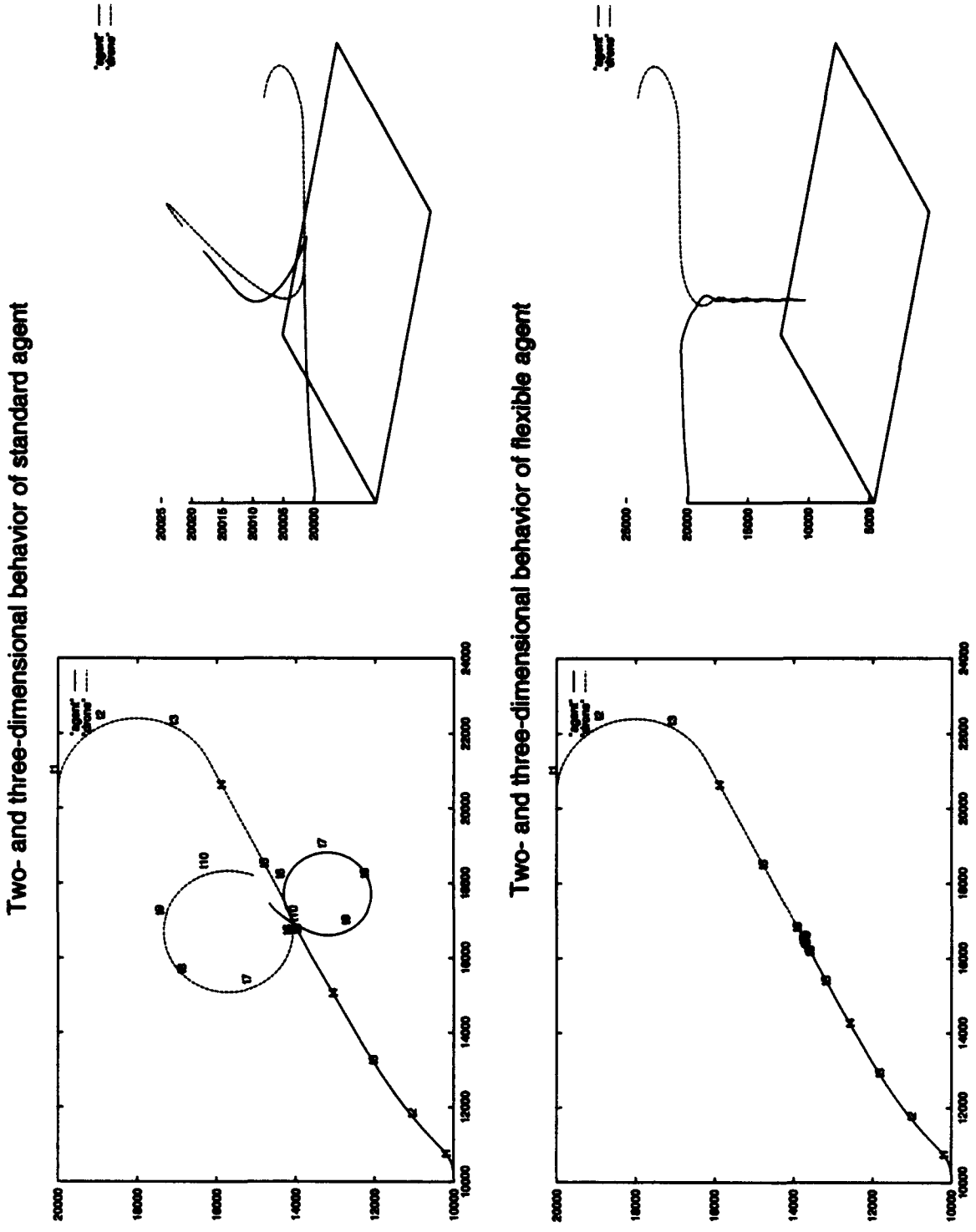

Figure B.9 **Scenario #1, long-distance range, agent moving slower than target.**

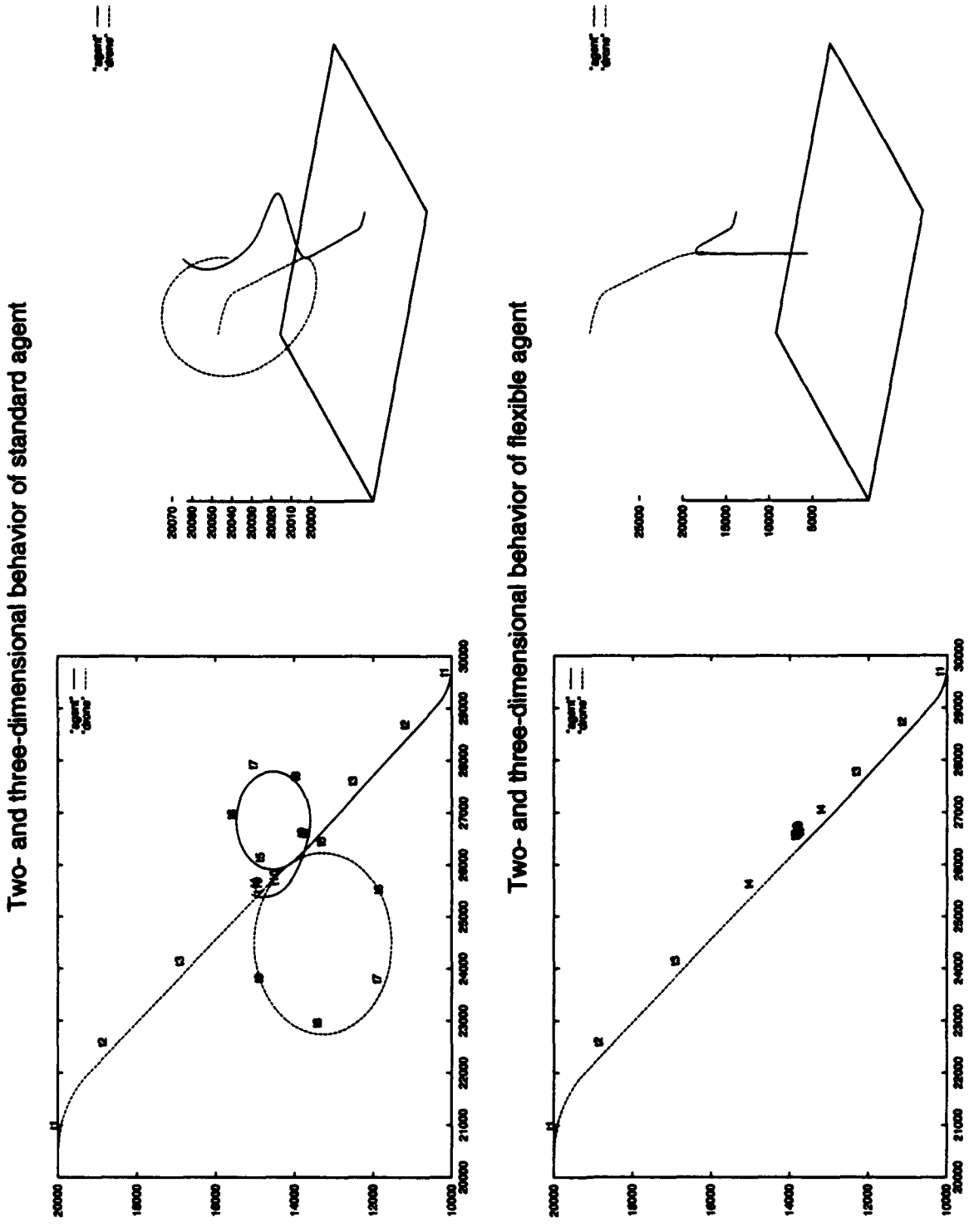

Figure B.10 **Scenario #2, long-distance range, agent moving slower than target.**

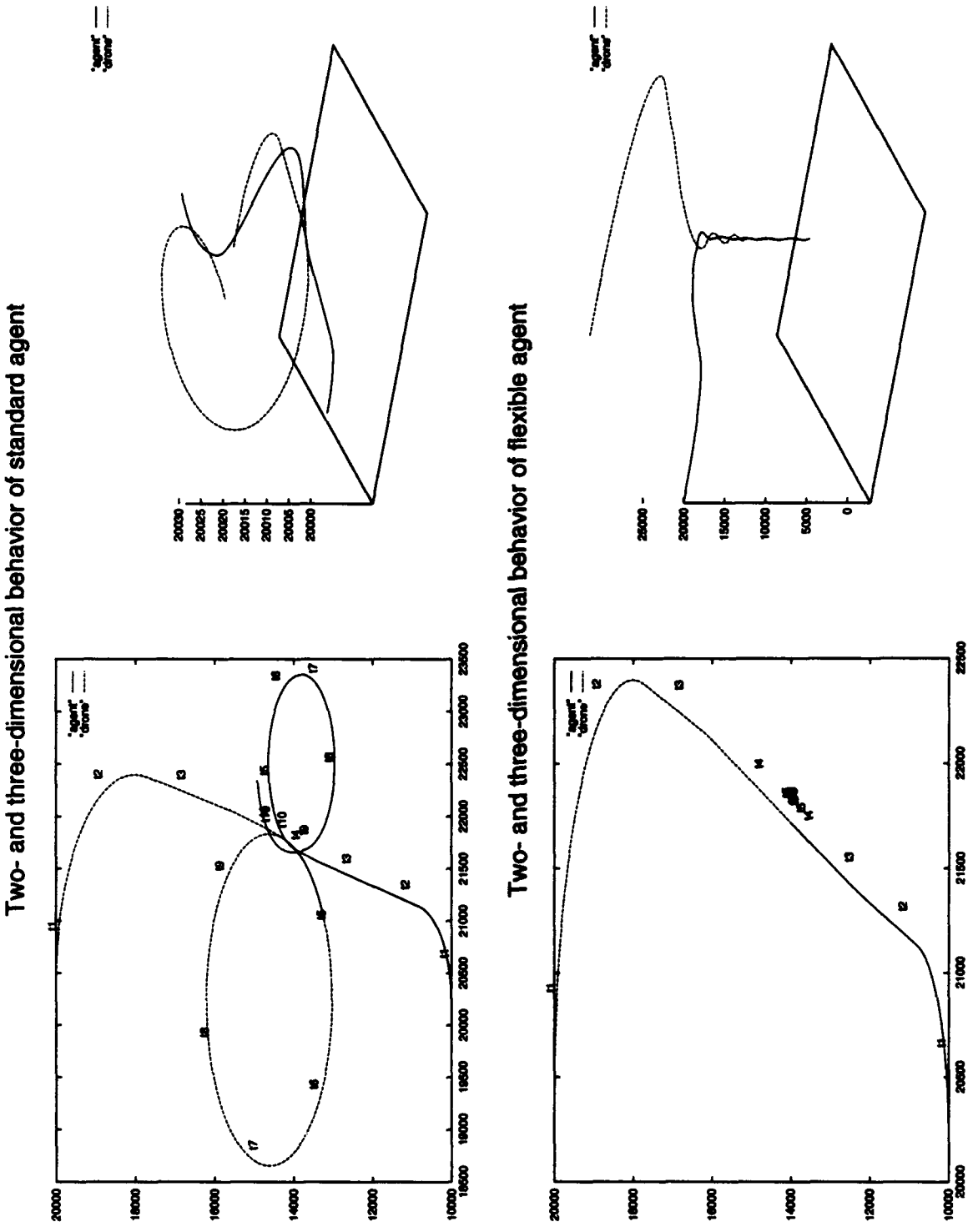

Figure B.11 **Scenario #3, long-distance range, agent moving slower than target.**

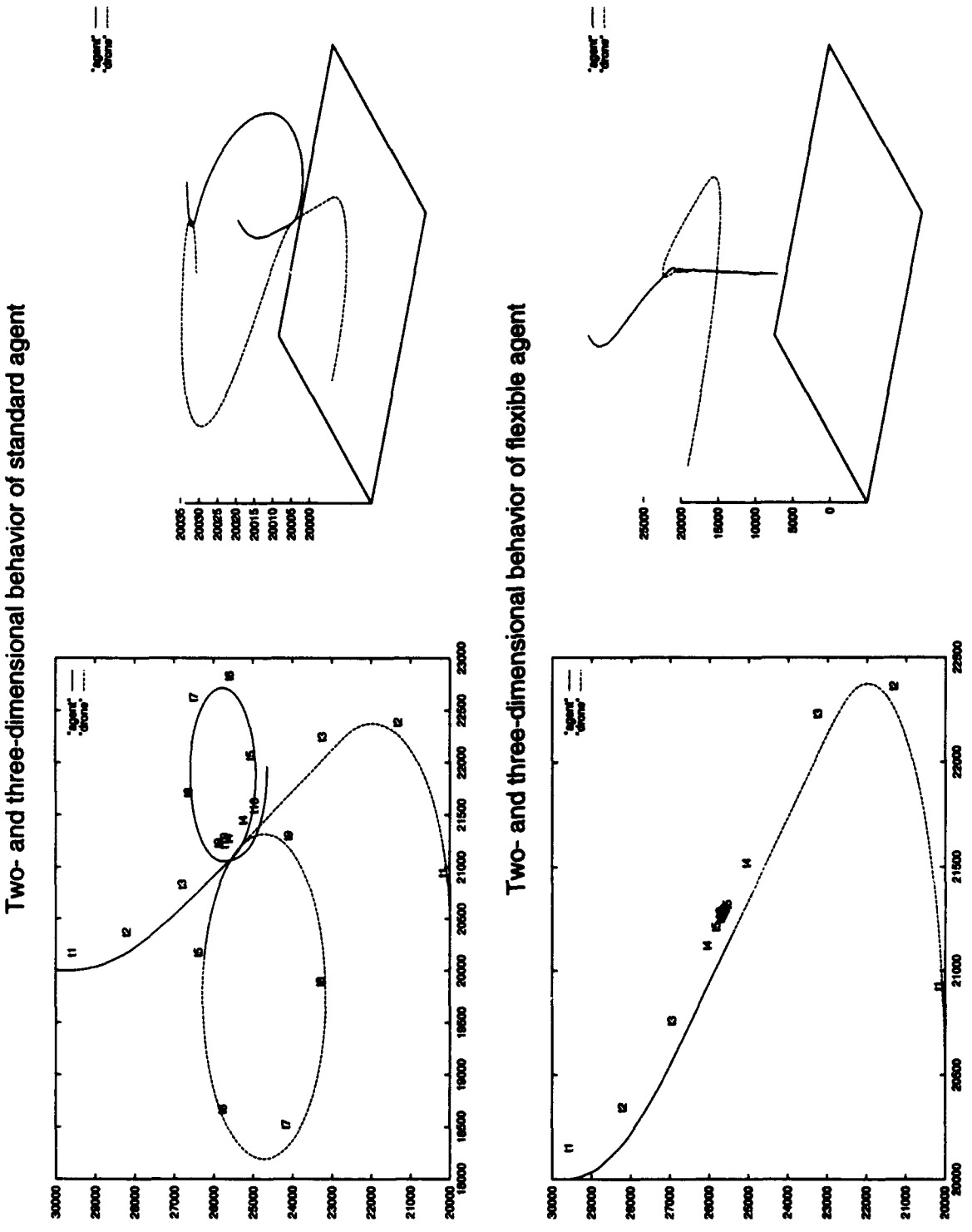

Figure B.12 **Scenario #4, long-distance range, agent moving slower than target.**

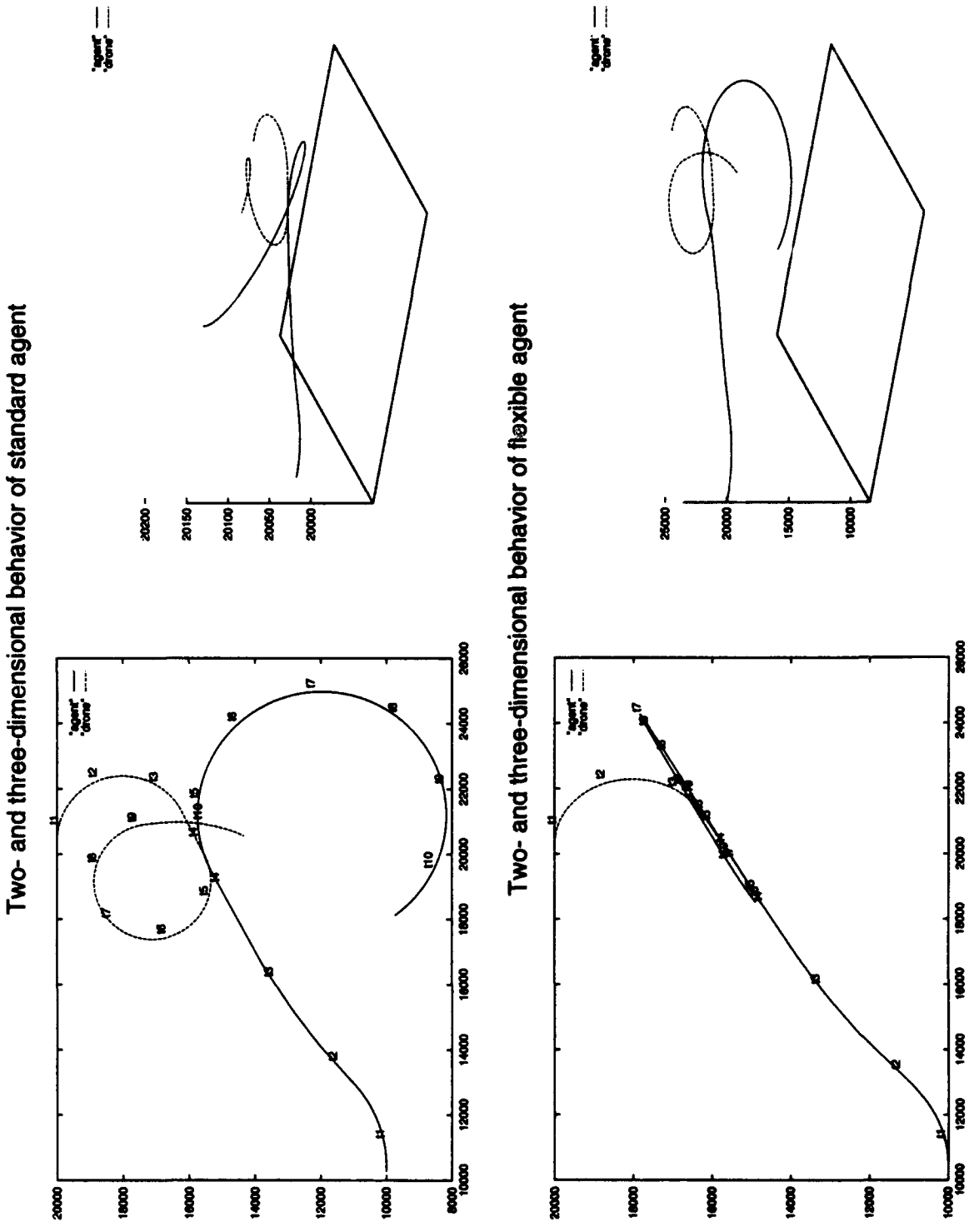

Figure B.13 **Scenario #1, long-distance range, agent moving faster than target.**

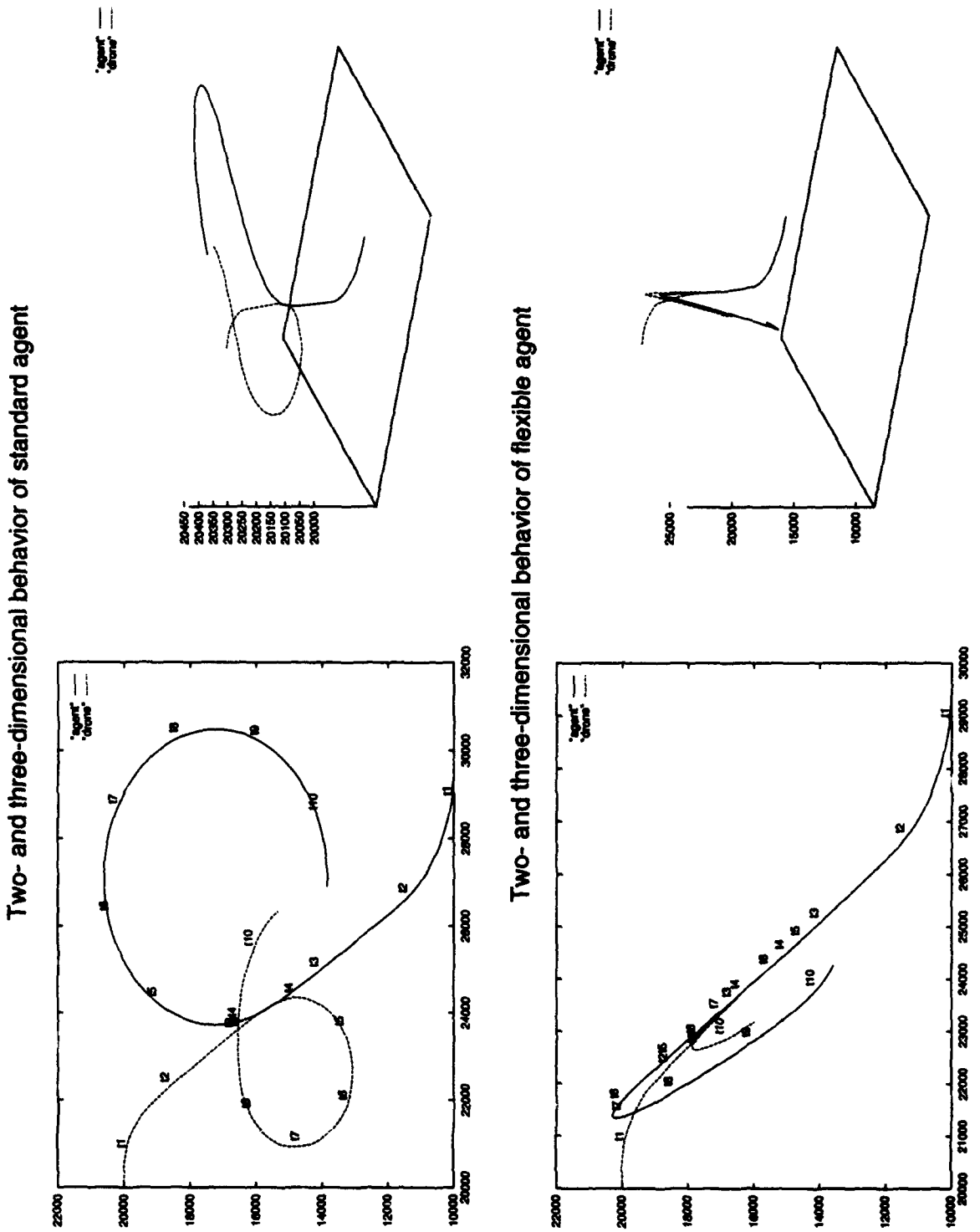

Figure B.14 **Scenario #2, long-distance range, agent moving faster than target.**

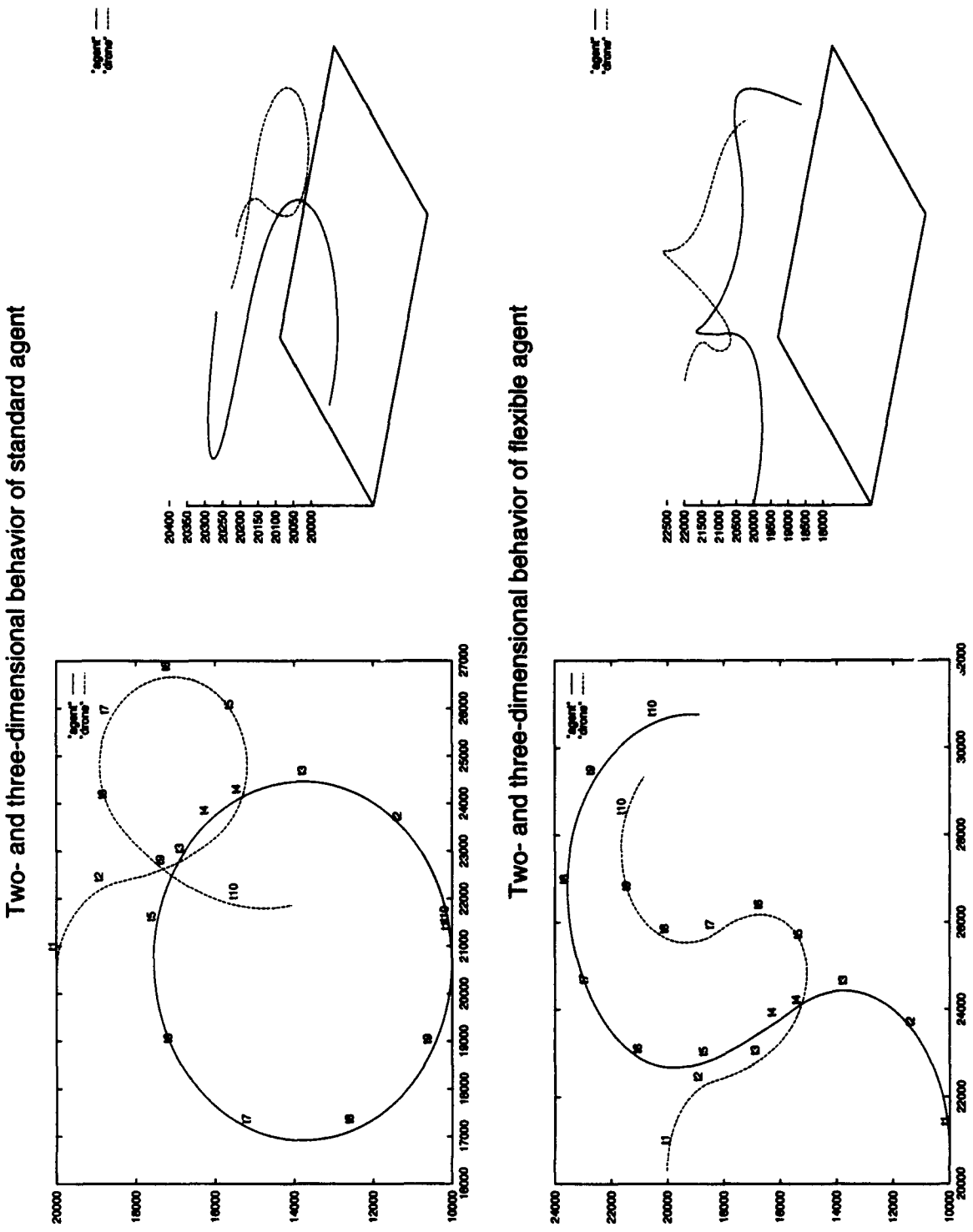

Figure B.15 **Scenario #3, long-distance range, agent moving faster than target.**

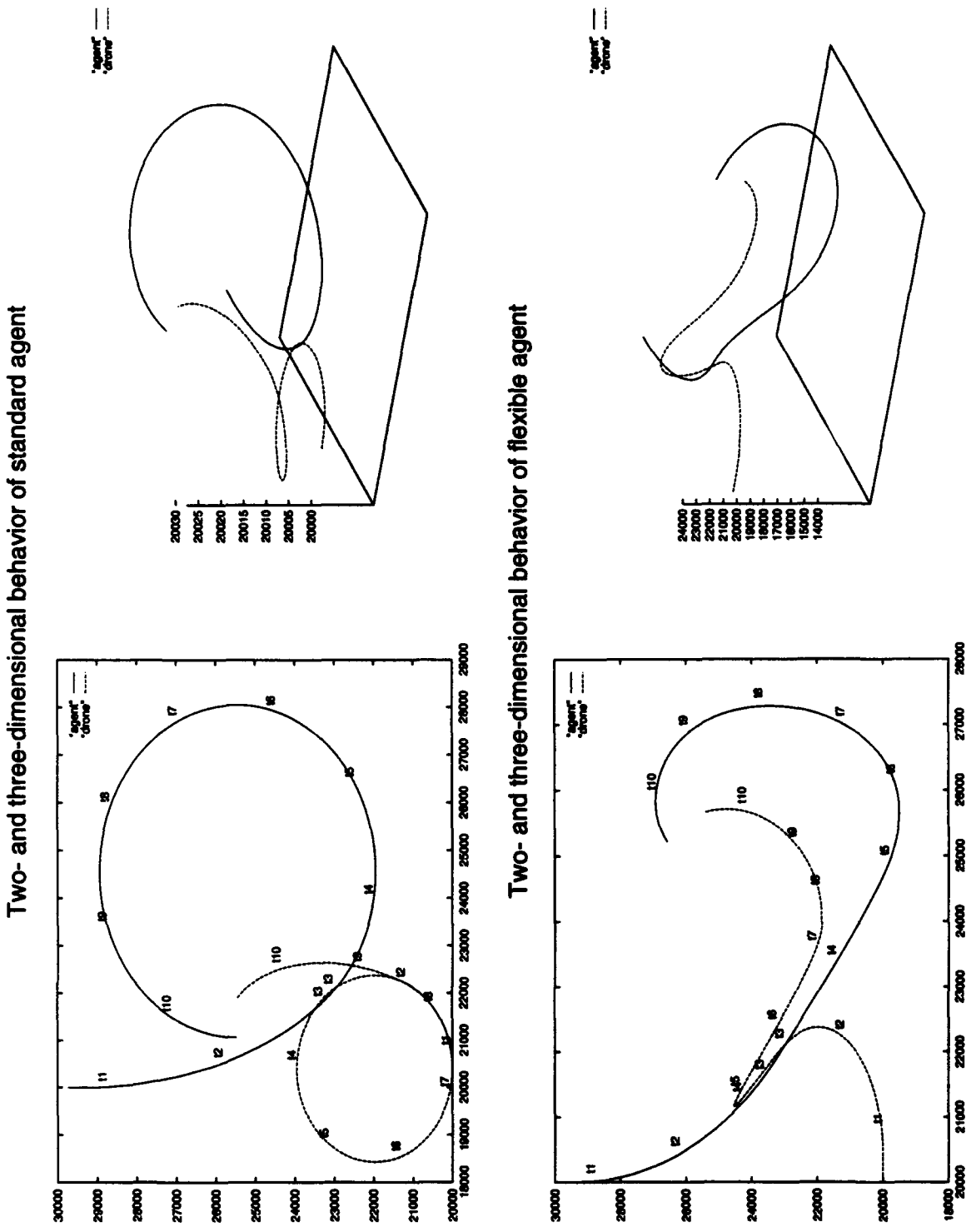

Figure B.16 **Scenario #4, long-distance range, agent moving faster than target.**

Appendix C. Source Code & Flow Charts

The source code for NOSTRUM is not included as part of this document. Two flow diagrams have been provided, however, to illustrate the main elements within the NOSTRUM program and how they interact with one another. Those interested in obtaining a copy of the source code should direct their requests to:

Maj. Gregg Gunsch (AI Lab)
AFIT/ENG
2950 P St.
WPAFB, OH 45433-7765
ggunsch@afit.af.mil

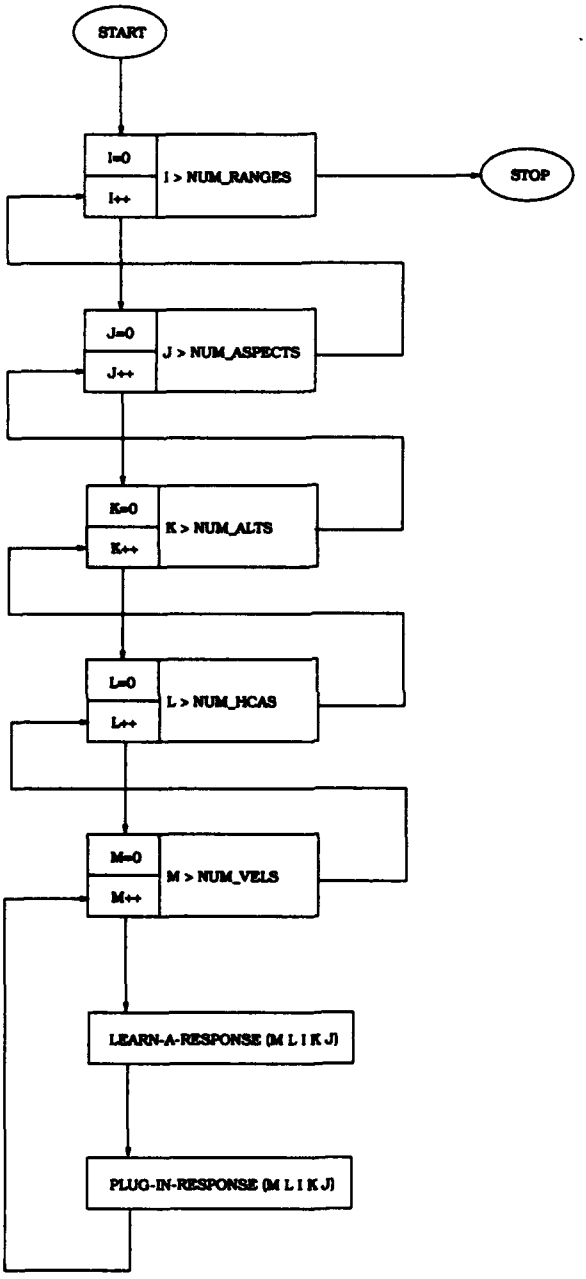

Figure C.1 NOSTRUM **main program loop.** This code repeats the learning loop for each of the 144 responses in the flexible agent universal plan.

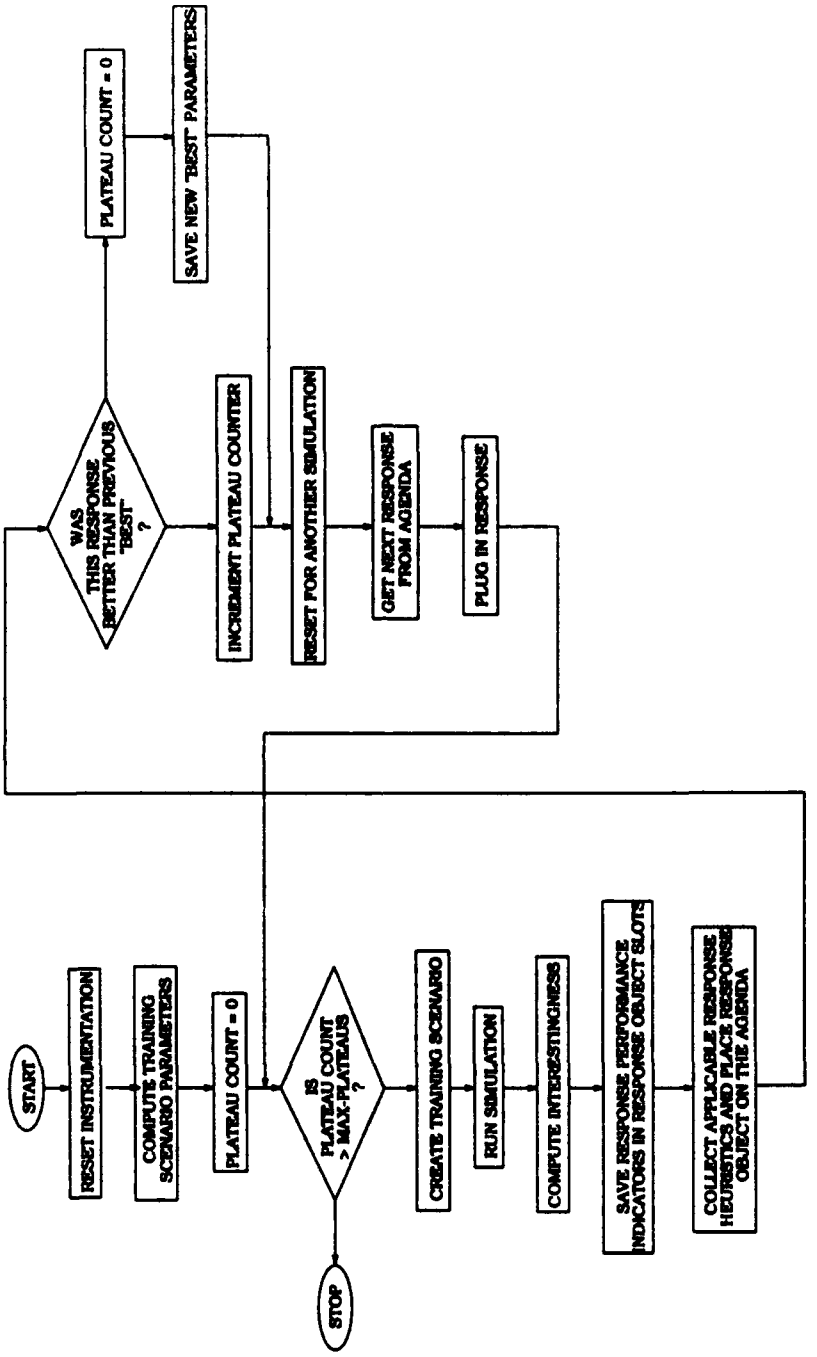

Figure C.2 **Sequence of events for each response learned.** This diagram describes the program flow as NOSTRUM explores alternative responses for a single plan sector.

Bibliography

1. Chaib-draa, B. and E. Paquet. "Integrating Reaction, Planning and Deliberation in Architecture for Multiagent Environment." *Proceedings of the Third Conference on Computer Generated Forces and Behavior Representation*. March 1993.
2. Cohen, Paul R. and Edward A. Feigenbaum. *The Handbook of Artificial Intelligence, Vol. 3*. William Kaufmann, Inc., 1982.
3. DeJong, Gerald and Raymond Mooney. "Explanation-Based Learning: An Alternative View," *Machine Learning*, *1*:145–176 (1986).
4. Dietterich, Thomas G. "Learning at the Knowledge Level," *Machine Learning*, *1*:287–316 (1986).
5. Dietterich, Thomas G. "Machine Learning," *Annual Review of Computer Science*, *4*:255–306 (1990).
6. Dyer, Douglas E. and Gregg H. Gunsch. "Enlarging the Universal Plan for Air Combat Adversaries." *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*. March 1993.
7. Gunsch, Gregg H., et al. "Autonomous Agents as Air Combat Simulation Adversaries." *Proceedings of the Eleventh Applications of Artificial Intelligence Conference*. April 1993.
8. Gunsch, Gregg H., et al. "On Applying Machine Learning to Air Combat Agents." *Fourth Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*. September 1993.
9. Hipwell, Dean P. *Developing Realistic Cooperative Behaviors for Autonomous Agents in Air Combat Simulation*. MS thesis, Air Force Institute of Technology, 1993. AFIT/GCE/ENG/93D-05.
10. Hluck, George S. *Developing Realistic Behaviors in Adversarial Agents for Air Combat Simulation*. MS thesis, Air Force Institute of Technology, 1993. AFIT/GCE/ENG/93D-06.
11. Jones, Randolph M., et al. "Intelligent Automated Agents for Flight Training Simulators." *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*. March 1993.
12. Katz, Amnon. "Intelligent Player - First Principle Foundations." *Proceedings of the Eleventh Applications of Artificial Intelligence Conference*. April 1993.
13. Kilpatrick, F. Alex. *An Investigation of Discovery-Based Learning in the Route Planning Domain*. MS thesis, Air Force Institute of Technology, 1992. AFIT/GCE/ENG/92D-07.
14. Kolcum, Edward H. "Gulf War Training Deficiencies to Dictate Future of Simulation," *Aviation Week and Space Technology*, *135*:51 (1991).
15. Laird, John, et al. *Soar User's Manual Version 5.2*. Technical Report CMU-CS-90-179, Carnegie-Mellon University, October 1990.

16. Laird, John E., et al. "Chunking in Soar: The Anatomy of a General Learning Mechanism," *Machine Learning*, *1*:11–46 (1986).

17. Langley, Pat, et al. *Scientific Discovery: Computation Explorations of the Creative Process*. MIT Press, 1987.

18. Langley, Pat, et al. "Heuristics for Empirical Discovery." *Computational Models of Learning* Berlin: Springer-Verlag, 1987.

19. Lenat, Douglas. "The Ubiquity of Discovery," *Artificial Intelligence*, *9*:257–285 (1977).

20. Lenat, Douglas. "The Nature of Heuristics," *Artificial Intelligence*, *19*:189–249 (1982).

21. Lenat, Douglas. "Theory Formulation by Heuristic Search," *Artificial Intelligence*, *21*:31–59 (1983).

22. Mezera, David. "Discussion of Flight Combat Domain With Pilot Capt Mike Gardner." Research notes, June 93.

23. Michalski, Ryszard S. "A Theory and Methodology of Inductive Learning," *Artificial Intelligence*, *20* (1983).

24. Rich, Elaine and Kevin Knight. *Artificial Intelligence*. McGraw-Hill, Inc., 1983.

25. Rosenbloom, Paul S., et al. "A Preliminary Analysis of the Soar Architecture as a Basis for General Intelligence," *Artificial Intelligence*, *47*:289–325 (1991).

26. Schoppers, Marcel J. "Universal Plans for Reactive Robots in Unpredictable Environments." *Proceedings of the International Joint Conference on Artificial Intelligence*. 1987.

27. Shavlik, Jude W. and Thomas G. Dietterich. *Readings in Machine Learning*. Morgan Kaufmann, Inc., 1990.

28. Shaw, Robert L. *Fighter Combat: Tactics and Maneuvering*. Naval Institute Press, 1985.

29. Simon, Herbert A. *Machine Learning: An Artificial Intelligence Approach*. Morgan Kaufmann, Inc., 1983.

30. UCF Institute for Simulation and Training, "Distributed Interactive Simulation, Operational Concept," 1992.

31. USAF, "F-4 pilot training document, Geometry analysis and cutoff attacks."

REPORT DOCUMENTATION PAGE

OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1993	Master's Thesis

4. TITLE AND SUBTITLE

USING DISCOVERY-BASED LEARNING TO IMPROVE THE BEHAVIOR OF AN AUTONOMOUS AGENT

5. FUNDING NUMBERS

6. AUTHOR(S)

David N. Mezera, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCE/ENG/93D-10

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Computer-generated autonomous agents in simulation often behave predictably and unrealistically. These characteristics make them easy to spot and exploit by human participants in the simulation, when we would prefer the behavior of the agent to be indistinguishable from human behavior. An improvement in behavior might be possible by enlarging the library of responses, giving the agent a richer assortment of tactics to employ during a combat scenario. Machine learning offers an exciting alternative to constructing additional responses by hand by instead allowing the system to improve its own performance with experience.

This thesis presents NOSTRUM, a discovery-based learning (DBL) system designed to work in tandem with the MAXIM air combat simulator. Through a process of repeated experimentation modeled after the scientific method, NOSTRUM was able to discover many responses that were more appropriate than the single mode of agent control implemented in the original MAXIM program. NOSTRUM often found responses that dramatically improved the offensive position of the agent, and it sometimes placed the agent in position for an extended shot on the target when one was not available before.

14. SUBJECT TERMS	15. NUMBER OF PAGES
ARTIFICIAL INTELLIGENCE; MACHINE-LEARNING; DISCOVERY-BASED LEARNING; AIR COMBAT; SIMULATION	145
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL