Decentralized UAV guidance using modified boid algorithms

by

Jason Burdell Clark

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Aerospace Engineering

Program of Study Committee:
Ping Lu, Major Professor
Bion Pierson
Marzia Rosati

Iowa State University

Ames, Iowa

2004

Graduate College
Iowa State University

This is to certify that the master's thesis of

Jason Burdell Clark

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Decentralized guidance of Unoccupied Air Vehicles (UAVs) is a very challenging problem. Such technology can lead to improved safety, reduced cost, and improved mission efficiency. Only a few ideas for achieving decentralized guidance exist, the most effective being the boid algorithm. Boid algorithms are rule-based guidance methods derived from observations of animal swarms. In this paper, boid rules are used to autonomously control a group of UAVs in high-level transit simulations. This paper differs from previous work in that, as an alternative to using exponentially scaled behavior weightings, the weightings are computed off-line and scheduled according to a contingency management system. The motivation for this technique is to reduce the amount of on-line computation required by the flight system.

Many modifications to the basic boid algorithm are required in order to achieve a flightworthy design. These modifications include the ability to define flight areas, limit turning maneuvers in accordance with the aircraft dynamics, and produce intelligent waypoint paths. The use of a contingency management system is also a major modification to the boid algorithm.

A Simple Genetic Algorithm is used to partially optimize the behavior weightings of the boid algorithm. While a full optimization of all contingencies is not performed due to computation requirements, the framework for such a process is developed.

Wolfram's Matlab software is used to develop and simulate the boid guidance algorithm. The algorithm is interfaced with Cloud Cap Technology's Piccolo autopilot system for Hardware-in-the-Loop simulations. These high-fidelity simulations prove this technology is both feasible and practical. They also prove the boid guidance system developed herein is suitable for comprehensive flight testing.

CHAPTER 1. INTRODUCTION

Automation leads to increased productivity. This concept is the driving thought behind all research into advanced control systems. In the same manner, the ability for a single operator to manage multiple aircraft is a very useful achievement. Possible benefits of this technology include improved operator efficiency, the opportunity for distributed function, redundancy in hardware, reduced production and operating costs, and improved safety. However, the management of several aircraft simultaneously is wrought with challenges.

This research stems from a National Aeronautics and Space Administration (NASA) project under the Revolutionary Systems Concepts in Aeronautics (RSCA) program. The project, a collaboration between Dryden Flight Research Center (DFRC) in Edwards, California, and Ames Research Center (ARC) in Mountain View, California, was entitled Networked Unpiloted Air Vehicle Teams (NUAVT). The focus of the project was to show the feasibility of using cooperative unpiloted vehicles, both ground and air, to assist in forest fire fighting activities using Unoccupied Air Vehicles (UAVs). However, the technologies developed are general enough for use in many different applications.

In general, there are several different aspects of NUAVT vehicle operations. Transit consists of moving the vehicles from one location to another. Searching is the methodical exploration of a designated area using any of a number of methods. Mustering is the gathering of vehicles in a particular area before and after a search or transit. The final implementation of this project will show a seamless integration between these operations.

This thesis is primarily concerned with the transit phase of operations. A comprehensive boid algorithm will be used for moving vehicles in a distributed manner. In addition, since NUAVT involved the use of two vehicles, this paper will mostly investigate the simulations of only two vehicles. However, there will be some investigation of emergent orbit behaviors and tests with

multiple simulated vehicles.

1.1 Aim

Most previous work in automatic guidance is based upon some type of leader-follower technique [12], [13]. These techniques have demonstrated success with small numbers of aircraft but do not scale well to large numbers of aircraft.

While a great deal of research has been applied to formation flight of UAVs, there are few known cases of decentralized guidance applied to aircraft. Although some decentralized guidance techniques have seen basic flight tests, very few comprehensive projects have flown. Previous research on decentralized control for UAVs has focused mostly on simulation, such as Nash Bargaining techniques by Castillo [5] and Inalhan [17], or Mixed-Integer Linear Programming (MILP) techniques by Bellinghman [3] and Sigurd [27]. Flight tests of MILP algorithms have taken place using both a single UAV [29] and multiple UAVs [16]. However, many decentralized techniques are either too costly in terms of computation requirements, or otherwise impractical for flight in their current forms.

Boid algorithms are rule-based guidance methods derived from observations of animal swarms. Boid algorithms have shown a great deal of success in coordinating large numbers of simulated aircraft in a decentralized manner as evidenced by Reynolds [23], Crowther [7], and Park [21]. There are no known cases of flight tests involving aircraft using boid rules for guidance, and there is no known concerted effort to utilize boid rules for guidance in a flight-ready system. Such a design takes into consideration all the safety, operations, and mission concerns required for a complete test.

The aim of this thesis is to apply boid guidance rules to UAVs in a realizable flight design. Schedule and funding prevented the actual flight tests of the system before the writing of this thesis, but high-fidelity software and Hardware-in-the-Loop simulations will be utilized as the best alternative.

Computing power aboard small UAVs is very limited. In order to fly aircraft of this nature, the control and guidance systems must be kept concise. A secondary goal of this thesis is to minimize the required on-line computation wherever possible.

1.2 Approach

This paper focuses on the use of fixed-wing UAVs as opposed to ground vehicles or Vertical/Short Take-Off and Landing (V/STOL) aircraft. Simulation, both software and Hardware-in-the-Loop, was used to verify the control algorithms. The simulations used the parameters of the APV-3 aircraft, made by RnR Products, as this aircraft model will be used in future flight tests to be conducted at NASA DFRC. The Piccolo® autopilot system, developed by Cloud Cap Technology in Hood River, Oregon, was used to autonomously control the aircraft [28].

Using a robust and thoroughly verified autopilot was a critical part of this research project. The highly capable Piccolo system manages the aircraft stability and control, allowing the research to focus on the aircraft guidance. Use of the Piccolo system does present several challenges, however, including the need to design flight paths using waypoint guidance. The Piccolo avionics hardware also has very little storage and computation capacity, requiring the development of systems with very minimal on-line computation requirements.

Boid algorithms were chosen for their ease of implementation and distributed nature. Using boid rules, there is no inherent need for a leader or a centralized controller (although a ground station was used in this implentation).

Wolfram's Matlab® (a powerful commercial matrix manipulation program) was used to model and simulate the aircraft guidance systems and the boid algorithms. This software tool has been used for many control and guidance problems with a great deal of success.

The resultant boid guidance system required optimization. Due to the nature of the boid algorithm, optimization is a difficult task. Previous work by Goldberg [11] showed success using Genetic Algorithms for optimization of complex functions, and Dimock [9] showed success with boid algorithms in particular.

1.3 Overview

This paper is organized as follows: Chapters 2 and 3 give background information relevant to this thesis; Chapters 4–6 describe the application of the needed techniques and tools; and Chapters 7 and 8 present and analyze the results.

Chapter 2 consists of detailed background information about research into cooperation among

aircraft. This chapter discusses two types of cooperative aircraft guidance (formation guidance and decentralized guidance) and the basic implementations of each type.

Boid algorithms are introduced in Chapter 3. The important elements and applications of boids are discussed in this chapter.

Chapter 4 explores the application of boid rules to aircraft. Specifically, it explains the adaptation and selection of the rules, determination of behavior weightings, and contingency management.

Genetic Algorithms are the focus of Chapter 5. A complete synopsis of the basis for Genetic Algorithms and an explanation of how they are used to optimize boid algorithms are given.

The simulation environment is discussed in Chapter 6. This includes the optimization and the software simulation of the boids. Also covered is the hardware-in-the-loop simulation provided with the Cloud Cap Piccolo system and the modifications to it.

The simulation results are reported and analyzed in Chapter 7. Also discussed are the results from the optimization.

Chapter 8 pulls together the results discussed in the preceding chapters and identifies the implications and the opportunities for future work.

1.4 Outcomes

The major contribution of this thesis is the development of a flightworthy distributed guidance system using boid rules, and a design methodology for future implementations. Many modifications to the boid algorithm were also made in the development of this thesis.

The use of Genetic Algorithms for boid algorithm optimization is further validated with this work. In addition, the BackStep method, which may lead to faster Genetic Algorithm convergence is investigated.

The results of this research show that boid algorithms can be used to effectively and easily control the cooperative transit of fixed-wing UAVs. Hardware-in-the-Loop tests prove that this technology is both feasible and practical. This is also the first investigation of a contingency-based boid implementation and the first known application of boid algorithms to a waypoint-based guidance system.

Novel ideas include a boid-based thermal centering behavior, using boids for battlefield aerial skirmishing, and using boids for automated highway systems. The ability to direct an emergent orbit behavior is also novel.

Important tools for the further study and implementation of boid rules were developed in the course of this study, including two Matlab graphical user interfaces (GUIs) for interacting with the simulation.

CHAPTER 2. BACKGROUND: FORMATION AND SWARM GUIDANCE

There are many different methods for managing multiple vehicles. The methods that fall under the category of formation guidance rely on assigning each vehicle specific positions within the group. These methods have an advantage in their simplicity but are often too rigid or not robust enough for some applications. A decentralized guidance scheme employs a distributed framework for manipulating the motions of the swarm. sDecentralized guidance methods are not concerned with specific vehicle positions but rather with the mission objectives. Decentralized methods can be very effective but are often extremely complex to implement and computationally expensive. Boid algorithms are a type of decentralized guidance that use simple interaction rules to achieve mission objectives, and will be discussed in Chapter 3.

2.1 Guidance and Control

In classical terms, control refers to the manipulation of system dynamics, be it deflecting an aileron to achieve a turn, closing a valve to reduce fluid flow, or translating a piston to achieve compression. A control system is often used to produce an artificially stable platform [4].

In contrast, a guidance system establishes and controls the desired flight path [26]. Guidance is the process of commanding various activites such as steering, velocity, and heading angle. An aircraft guidance system relies on measurements of position, velocity, and acceleration in order to generate and maintain flight paths. This is in contrast to control systems which may require many additional measurements such as surface positions, turning rates, or air temperature.

To generate realistic guidance paths for an aircraft, care must be taken such that the commanded trajectories do not violate the physical constraints of the system. For an aircraft, this includes not exceeding the maximum speed, not falling below the stall speed, and not making turns tighter than the minimum turn radius. If these limits are not considered, the aircraft will

not be able to execute the desired guidance commands.

Both guidance and control are required for a successful aircraft system. An aircraft is useless if it does not maneuver and movement is pointless if it can not be controlled. This paper is solely interested in the guidance of multiple aircraft and treats it independently from whatever control system may be employed.

2.2 Formation Guidance

2.2.1 Leader-Follower Guidance

In Leader-Follower guidance, one vehicle is designated as the leader. Each vehicle follows the previous one in line in a serial "daisy-chain" fashion. This method proved successful when used with a small number of aircraft [13].

Leader-Follower guidance is easy to implement and requires very little on-line computation. However, the formation geometry must be pre-defined and each vehicle must be explicitly assigned a position. Also, the disturbance rejection of these guidance systems can be very poor. If the formation leader is disturbed, all of the follower vehicles track the resultant trajectory.

A simple model of a Leader-Follower system was contructed for demonstration using Matlab's Simulink toolbox. Using a dynamic model derived from linearizing the full aircraft Six Degrees-of-Freedom (6-DoF) equations about a steady level flight condition, a standard linear quadratic regulator (LQR) controller was constructed. Each simulated aircraft used a common plant and controller. The inputs to the controller are position and velocity commands. The position commands are hard-coded for each aircraft; that is, each aircraft has a predefined position in the formation. The lead aircraft receives the velocity command to the formation. The resulting position and velocity of the leader are then "sensed" by the first follower aircraft and used as inputs. The second follower aircraft senses the first follower, and so forth.

The velocity response for a four-aircraft formation using Leader-Follower guidance is shown in Figure 2.1. Here, a step velocity command is issued to the leader aircraft at t=10 seconds, followed by a ramp wind disturbance to the leader at t=16 seconds. The relative position response is shown in Figure 2.2. As seen in the figures, the errors propogate to the follower aircraft, with the disturbance increases for each aircraft in line.

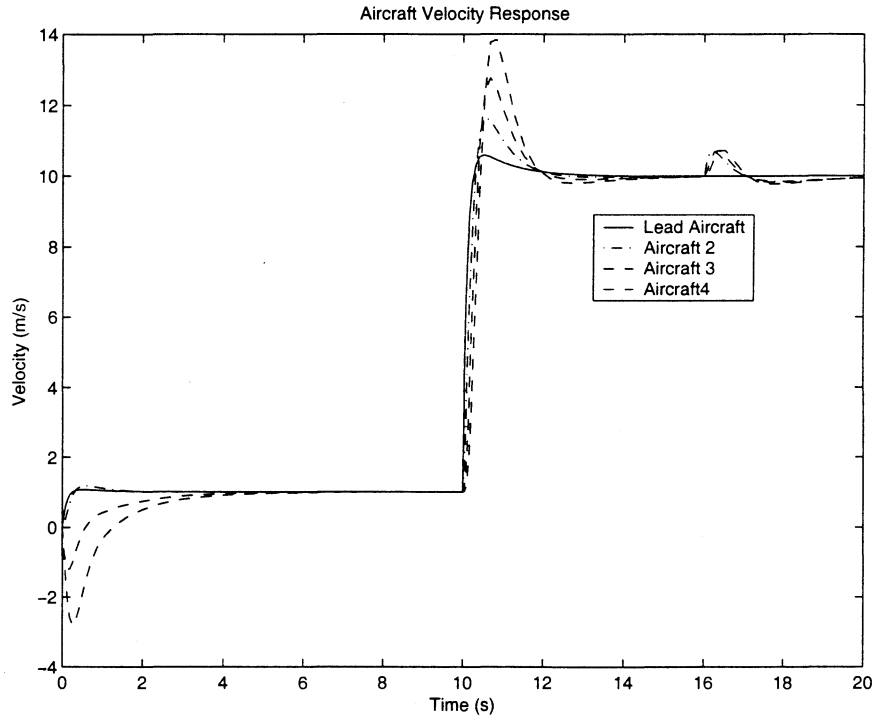

Figure 2.1 Velocity response for Leader-Follower guidance

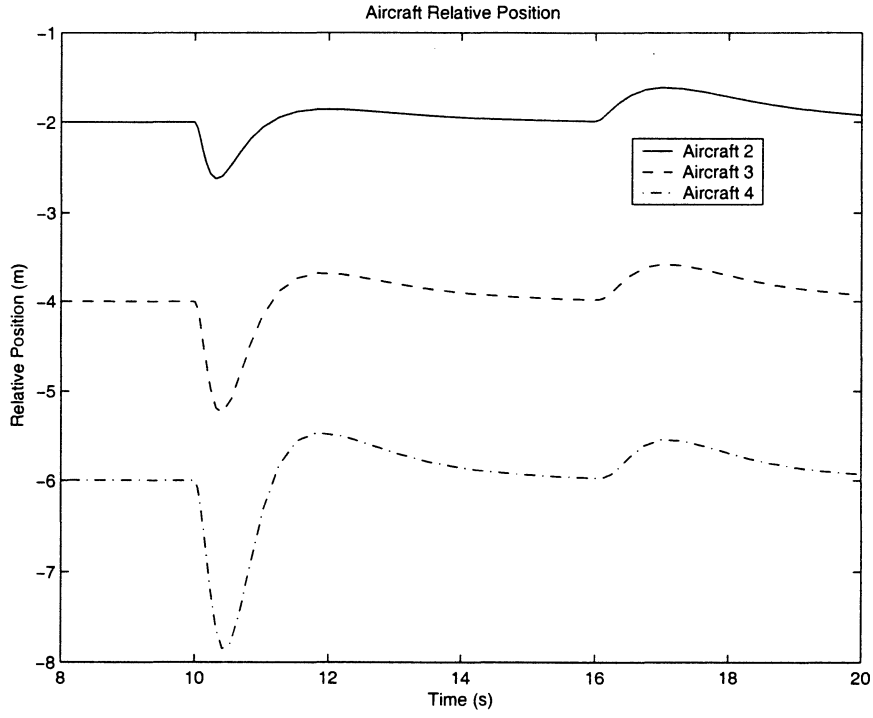

Figure 2.2 Position response for Leader-Follower guidance

2.2.2 Virtual Leader Guidance

Adding a virtual leader to the Leader-Follower guidance system serves to improve its disturbance response. In this implementation, an imaginary vehicle is projected into the control space and made to follow the desired trajectory path. Since the vehicle is virtual, it is not affected by disturbances. The disturbances therefore do not propogate to the followers. Giulietti, *et al* demonstrated the effectiveness of this technique in simulation [12].

Just as with the Leader-Follower Guidance, a model of the Virtual Leader Guidance system was created for demonstration. The basic model is very similar to the Leader-Follower model, except that all aircraft feed off the position and velocity of the virtual leader.

The velocity response for a three-aircraft formation using a virtual leader is shown in Figure 2.3 and the relative position response is shown in Figure 2.4. A step velocity command is given to the virtual leader at t=10 seconds and the first follower aircraft is disturbed by a ramp wind function at t=16 seconds.

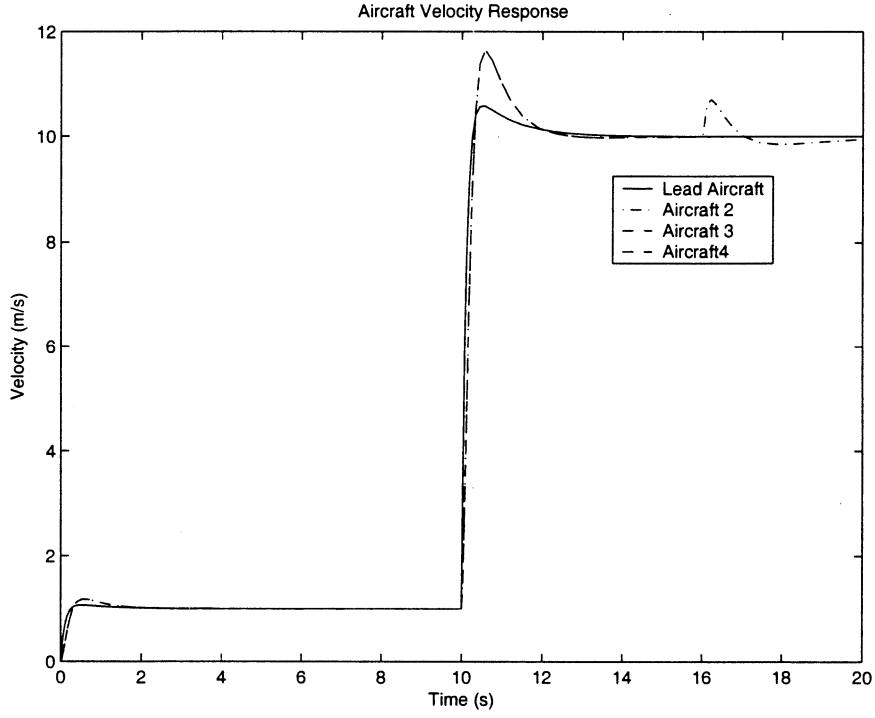

Figure 2.3 Velocity response for Virtual Leader guidance

As demonstrated here, formation guidance using a Virtual Leader is much more robust than

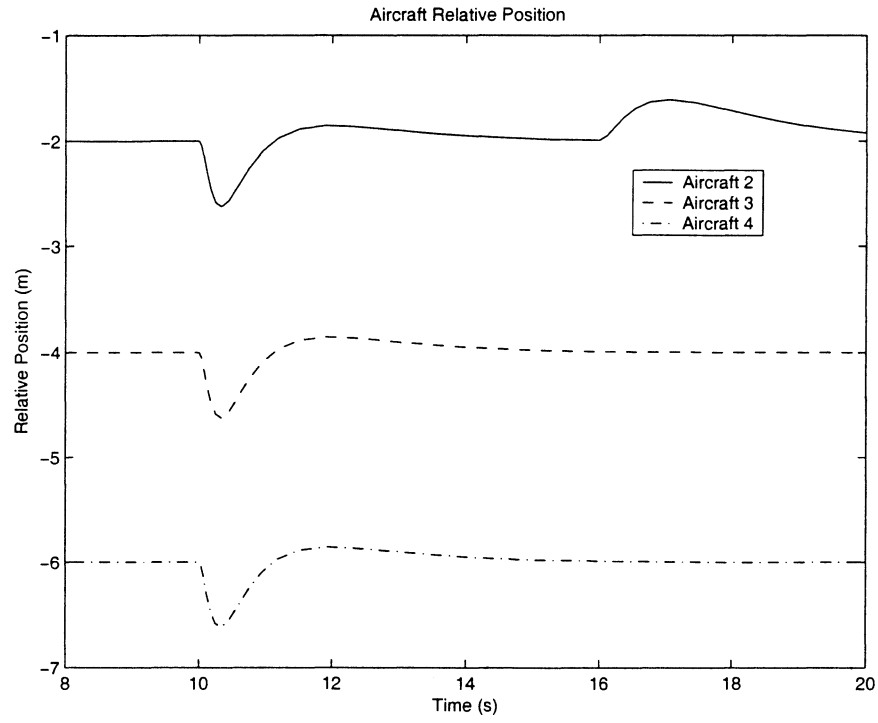

Figure 2.4 Position response for Virtual Leader guidance

the Leader-Follower method. For many applications, such as wingtip vortex utilization, the Virtual Leader method is ideal. However, this method does not address concerns such as obstacle avoidance or target tracking. Virtual Leader methods also require explicit determination of the formation geometry and the position for each aircraft in the group.

2.3 Decentralized Guidance

Few proposed methods exist for the decentralized guidance of aircraft. Two such decentralized methods are Mixed-Integer Linear Programming (MILP) solution techniques and Nash Bargaining solutions. However, as will be discussed, neither of these methods address the problem of autonomously moving a swarm of vehicles from one area to another. In addition, both require intensive on-line computation.

2.3.1 Mixed-Integer Linear Programming

MILP is a method for finding the optimal solution to a problem with continuous variables and variables that can only take integer values, given certain solution constraints. This is a very specific type of "choice" problem that can not be solved by traditional optimization methods. (Optimization methods will be discussed in Chapter 5.)

A classic example of a MILP problem is "The 0-1 Knapsack Problem." In this problem, a hiker must pack a knapsack with a subset of available items. Each item has a certain utility value associated with it as well as a weight. Since the hiker can only carry a certain amount of weight, the problem becomes optimizing the total utility in the knapsack without exceeding the weight limit [5].

Aircraft guidance techniques involving MILP yield optimized transit paths for multiple aircraft given certain mission constraints. The MILP problem formulation is not necessarily intended for transiting a swarm from one place to another. Rather, the problem uses multiple types of vehicles and multiple types of waypoints that must be visited. Certain vehicles are more suitable for some of the waypoint types. In addition, the vehicles have "no-fly-zones" (obstacles) that must be avoided.

Richards showed that the task assignment and trajectory planning problems can be combined into one MILP optimization problem [24]. Richards uses contraint equations to ensure each waypoint is visited exactly once by a vehicle with suitable capabilities, to ensure each aircraft avoids rectangular no-fly regions, to limit velocity, and to limit the turn rate. The optimization objective is simply to minimize the total mission time.

The research by Richards yields optimized paths for each aircraft, which may be uploaded to each aircraft. The aircraft act independently from one another, achieving the goal of decentralized guidance.

There are some drawbacks to MILP techniques, however. Solution of MILP problems requires intensive on-line computation. Even very simple problems take several minutes to solve using high-end desktop computers [27]. An approximation method did reduce this computation time to seconds, but at the cost of reduced precision. In addition, the algorithm must design the complete trajectories for the mission and the problem must be solved by a centralized computer. This is a

less than ideal situation.

A major concern is that MILP formulations are not designed to produce trajectories which keep the aircraft close together or heading in the same direction. In this respect, MILP methods are, in fact, solving problems significantly different from the stated goals of this paper.

2.3.2 Nash Bargaining

Nash Bargaining techniques are algorithms borrowed from the world of finance and applied to conflict resolution in aircraft. John Nash won the Nobel Prize in 1994 for his 1950 paper entitled "The Bargaining Problem" [20] which presents a simple, yet elegant solution to the problem of competing bargainers.

The basic theory behind Nash Bargaining is that in order to obtain a globally optimal solution, each bargainer is required to make proportional sacrifices. This precludes the need to analyze the bargaining process itself, and instead allows the analyst to focus on specifying the properties that a reasonable solution must possess.

Nash Bargaining assumes the decision-makers are highly rational, are equal in bargaining skill, and have full knowledge of the desires of their opponents. For a computerized bargaining system, these appear to be reasonable assumptions. Formally, the key axioms proposed by Nash are:

1. Any agreement between the bargainers must be within the bargaining set.
2. Both bargainers are equal in bargaining power.
3. The "measuring stick" used in assesing utility values is irrelevant.
4. Irrelevant solution alternatives are ignored. (Irrelevant alternatives are defined such that if solution s is preferred when t is available, t is an irrelevant alternative.)

The Nash solution is guaranteed to be efficient (satisfies Pareto optimality), rational, and feasible [25]. Optimal efficiency is very attractive for use with aircraft because resources such as fuel are very limited.

Some researchers have investigated using Nash Bargaining for aircraft conflict resolution. The research by Inalhan, *et al* [17] links the decentralized aircraft problem to the centralized optimization problem. This finding removes the need for a central computer to obtain the optimized

solution. The proofs given by Inalhan show that the decentralized solution will be indistinguishable from the centralized one. Thus, using Nash Bargaining for aircraft guidance has advantages in that no centralized coordinator is required. The aircraft interact with their nearby counterparts and barter for a solution.

Some major drawbacks to using Nash Bargaining algorithms are the on-board computation requirements and the need for calculation of large sections of the paths. It is also required that the vehicles be able to directly communicate with one another, although imperfect communication scenarios are allowed.

Nash Bargaining is mostly designed for conflict resolution between aircraft. It does not include provisions for flocking, velocity matching, target seeking or obstacle avoidance. Therefore, it is clearly not yet adaptable to full flock transit problems.

CHAPTER 3. BACKGROUND: BOID ALGORITHMS

As stated in Chapter 1, boid algorithms are rule-based guidance methods derived from observations of animal swarms. The term "boid" is derived from the concatenation of "bird android" and a single autonomous agent may be referred to as a "boid". Boid algorithms have their roots in the biological sciences, where researchers studied the complex cooperative behaviors of animals such as flocking birds, schooling fish, and swarming ants. It was proposed that these complex emergent behaviors could be explained if each animal agent were to follow a set of very simple rules. The combination of these very simple rules can lead to seemingly intelligent behavior.

The initial work by Reynolds [23] was intended to produce more life-like motions of birds and other creatures for computer games and motion pictures. The potential application to aircraft guidance was realized shortly after publication of that paper. This application of boids has since spawned a good deal of research as evidenced by Crowther [7], Giulietti, et. al. [12], and Park, et. al [21].

The draw of boid algorithms is in their simplicity. With very little direct input, boids will exhibit complicated behaviors. This can also be the curse of boids, however, as these resulting behaviors can often be unpredictable. As Reynolds put it, "these darn boids have a mind of their own!"

3.1 Basic Boid Rules

For each of the rules, or behaviors, a vector acceleration command is calculated. These acceleration commands must be combined into a total acceleration. This is accomplished by applying a weight (gain) value to each acceleration command and finding the sum. This process will be discussed in detail in Chapter 4. These unit acceleration values are then multiplied by a scale factor for translation to the actual physical acceleration. Boid algorithms do not necessarily

predict an entire flight plan, but instead calculate moment to moment acceleration adjustments based on the behavior weightings.

The most basic boid algorithm is composed of three rules. The first of these rules provides that each boid will steer to align its heading with that of the group, or swarm. The second is to steer towards the center of the swarm. The third rule commands the boid to move away from the nearest neighbor boids. These behaviors are shown graphically in Figures 3.1, 3.2, and 3.3 [21]. In these figures, the circle represents the observable flock "neighborhood"; the white triangle represents the boid in question; the black triangles represent the detected neighbor boids; the bold arrows represent the resultant acceleration command; and the solid and dashed arrows represent the actual and desired velocity directions, respectively.

The velocity matching acceleration is calculated in Equation 3.1. This behavior commands the boid to move so that its velocity vector, $\vec{v}$, is pointing in the same direction as the velocity vector of its nearest neighbor, $\vec{v}_n$. Since the flock size is small for this implementation, the nearest neighbor is identified by a simple search of the entire flock.

$$\vec{a}_{match} = \frac{\vec{v}_{nearest} - \vec{v}}{\| \vec{v}_{nearest} - \vec{v} \|} \tag{3.1}$$

The geometric center of the flock, which is calculated in Equation 3.2, is used for finding the appropriate flocking acceleration. The position of each boid in the flock, $\vec{x}_i$, is averaged to obtain the geometric center of the flock, $\vec{x}_{center}$. The flocking behavior acceleration is calculated in Equation 3.3. For large swarms, it is more practical for each boid to calculate the geometric center of its nearest flock mates rather than that of the entire flock. Since this research involves small numbers of aircraft, that condition is relaxed.

$$\vec{x}_{center} = \frac{1}{n} \sum_{i=0}^{n} \vec{x}_i \tag{3.2}$$

$$\vec{a}_{flock} = \frac{\vec{x}_{center} - \vec{x}}{\| \vec{x}_{center} - \vec{x} \|} \tag{3.3}$$

The collision avoidance behavior uses only the distance to the nearest boid and generates an acceleration command in the opposite direction. This behavior is seen in Equation 3.4.

$$\vec{a}_{col} = -\frac{\vec{x}_{nearest} - \vec{x}}{\| \vec{x}_{nearest} - \vec{x} \|} \tag{3.4}$$

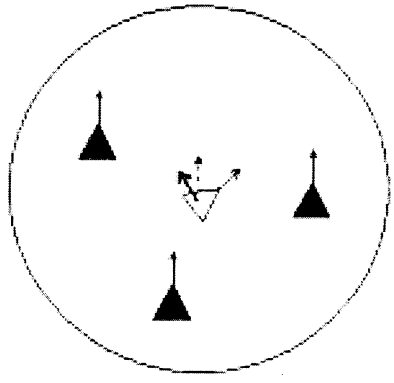

Figure 3.1 Velocity (heading) matching rule

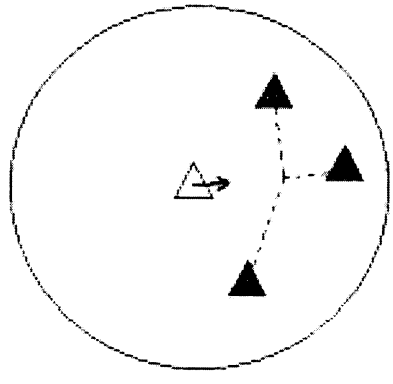

Figure 3.2 Flocking rule

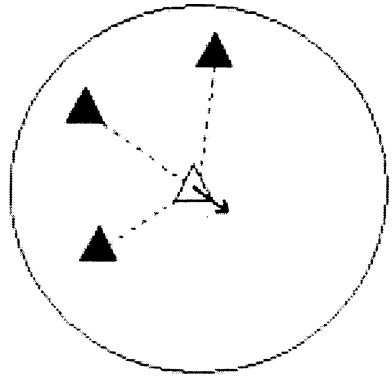

Figure 3.3 Collision avoidance rule

3.2 Supplementary Boid Rules

In addition to the three basic boid rules, there is any number of supplementary rules that can be used to help govern the emergent behavior. Some common supplementary rules include seeking a target, avoiding an obstacle, and pursuing or evading a moving target. Graphical representation of these behaviors can be seen in Figures 3.4 and 3.5 [21].

The seek behavior is implemented in a fashion similar to that of the flocking behavior, except there is no need to calculate any geometry since the final waypoint does not move. The seek calculation is shown in Equation 3.5.

$$\vec{a}_{seek} = \frac{\vec{x}_{waypoint} - \vec{x}}{\| \vec{x}_{waypoint} - \vec{x} \|} \tag{3.5}$$

The obstacle avoidance calculation is similar to the collision avoidance calculation. The boid is told to avoid the nearest obstacle as shown in Equation 3.6. The nearest obstacle is calculated using a method similar to finding the nearest boid.

$$\vec{a}_{obs} = -\frac{\vec{x}_{obstacle} - \vec{x}}{\| \vec{x}_{obstacle} - \vec{x} \|} \tag{3.6}$$

Pursuing a target or evading a moving obstacle are also very useful supplemental rules. A rough schematic of such behaviors is shown in Figure 3.5. Such rules work like a combination of the flocking and velocity matching rules, but applied to a single object rather than the neighboring boids.

Another rule that could be very useful for UAV operations is known as the arrival behavior. A representation of this behavior may be seen in Figure 3.6. Essentially, this behavior commands the boid to smoothly reduce speed as it approaches a target.

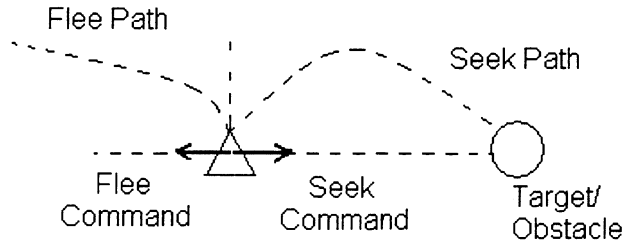

Figure 3.4 Seek/Flee behavior

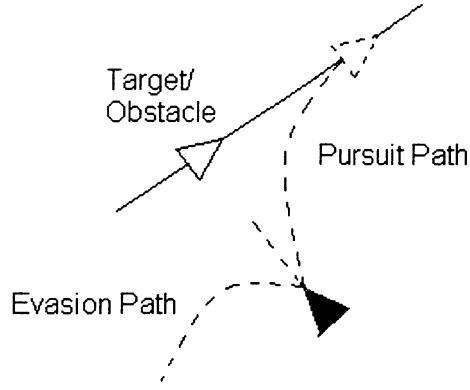

Figure 3.5 Pursue/Evade behavior

3.3 Advanced Boid Rules

Advanced boid rules are more complicated and detailed than supplementary boid rules. Rules of this type are typically application-specific. For instance, the drag reduction rule is mostly

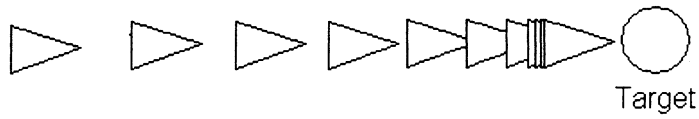

Figure 3.6 Arrival behavior

applicable for long-distance transit applications. The rules introduced in this section give only a sampling of the possible advanced boid rules.

3.3.1 Drag Reduction Rule

The drag reduction rule is an example of an advanced boid rule. Researched by Dimock, *et al* [9], the goal is to achieve global drag reduction by reducing local drag. This work was inspired by migrating bird flocks, which acquire a "V" formation in order to achieve energy efficiency [14]. Each boid is commanded to move along the local reduced drag gradient. In simulations, a flock of boids was started in a "V" formation. The flock remained in a "V" shape but adjusted the angle of the vertex according to the local optimization.

The drag reduction rule consists of commanding the boids to move in the direction of decreasing induced drag. It should be noted that this rule is calculated only for the spanwise direction, y, unlike the rules discussed in the previous sections. In Equation 3.7, $\vec{a}_d$ represents the drag gradient, computed by perturbing the induced drag (calculated in Equation 3.8), by $\pm\Delta y$ (a small distance in the spanwise direction). In addition, ρ is the air density, V is the freestream velocity, $\Gamma(y)$ is the vorticity distribution, and α_i is the angle of attack. The results of this research are encouraging, although further study is needed to determine if the flock will assume a "V" when started from random positions and orientations.

$$\vec{a}_D = -\frac{\vec{a}_d}{\| \vec{a}_d \|} \tag{3.7}$$

$$D_i = \rho V \sum_{j=1}^{k} \Gamma(y)\alpha_i \Delta y \tag{3.8}$$

3.3.2 Proposed Thermal Centering Rule Scheme

Proposed here by the author is a rule to guide boids into atmospheric thermals (columns of heated air with a strong updraft). The strong lift of thermals can be harnessed to achieve high energy savings and huge increases in endurance, a fact glider pilots acutely know.

Merely finding thermals can present a challenge, mainly because thermals are invisible to the naked eye. (A good infrared sensor on an aircraft could help mitigate this problem, but this will not be considered here as it is unclear if the state-of-the-art infrared sensor can be used in this manner.) Thermals are usually detected through measurement of vertical speed, but there are other sensor possibilities that could drastically improve detection and mapping results [22].

Upon detection of a thermal, the vehicle must turn to orbit the center of the thermal in order to fully utilize its lift. This introduces many problems such as determining the proper turn direction, the ideal turn radius, and the right orbit speed. In fact, as Reichmann discusses in his comprehensive book on soaring techniques, the rules for entering and utilizing thermals can be very complicated [22].

From the classical control sense, reliably detecting and utilizing thermals is essentially an intractable problem. The exhaustive work by Wharington [31] on this topic uses reinforcement learning [19], a broad class of techniques encompassing neural networks, to detect and center about thermals. It may be possible to simplify the technique such that the boid determines whether the lift is greater on the right or left, then turns to that side. If the vertical velocity increases, it would maintain those conditions until the ascent rate returns to zero.

Wharington investigated techniques similar to that described above without much success. It does appear, however, that this may have been due to intricacies of the vehicle control system such as phugoid characteristics. Regardless, this technique will most likely result in a significantly reduced thermal centring success rate as compared to the work of Wharington. However, if the object of the rule is merely to gain a little bit of extra endurance, rather than maximizing endurance by fully utilizing every detected thermal, this rule could be fully adequate while maintining the minimal on-line computation characteristics of boid algorithms.

There are also detailed rules laid out by Reichmann that may be more fitting for boid algorithms and more effective than the basic technique described above. Some of these rules use fuzzy

parameters that would need to be carefully evaluated.

The likelihood of a thermal being utilized by more than one boid increases when using the other boid rules in conjuction with thermal centering rule. This is because the flocking rule will draw the other boids closer to the one that has found the thermal. There could be some complications with the velocity matching rule if the other boids try to match the heading of the spiraling boid. Clearly, there are many dimensions to this problem.

The ability to circle thermals could be particularly useful for operations near wildfires (the impetus for this research) where updrafts from burning fuel are very strong. There are many potential benefits from this proposed rule, and it certainly deserves further study.

3.4 Boid Applications

The potential applications for boid algorithms are almost limitless. Any instance where the desire is to move many vehicles from one point to another qualifies as a potential case.

3.4.1 Forest Fire Fighting

An excellent example of where boid rules can be effective is in supporting forest fire fighting activities. In general, UAVs are very useful in this area for actively searching for fires, monitoring fires, serving as communications relays, delivering supplies, or applying fire retardant. For a swarm of UAVs to operate in this environment, they must be able to transit from area to area cooperatively, and have the ability to search cooperatively.

Boid algorithms can be integrated into a search algorithm using a simple method. When no obstacle or vehicle collisions are imminent, the search algorithm runs normally. However, in the event of a safety hazard, the boid rules (specifically the anti-collision rules) will engage until the threat passes. This simple modification to any search algorithm adds a great deal of robustness.

Aircraft may be required to transit from the launch point to a wildfire area some distance away. Due to the remote nature of most fire-prone areas and the inherent urgency in fighting fires, the use of a robust, fast, and easy-to-use guidance system is key. For this reason, boid algorithms are an ideal choice for such an application.

The ability for an aircraft to autonomously orbit a location, be it stranded fire fighters or a

newly detected fire, is also a very important behavior for fire fighting support. Such an emergent behavior is well within the capabilities of boid algorithms and will be demonstrated later in this paper.

Aerial fire fighting activities at night are significantly reduced due to the danger to the pilots and the difficulty of performing such activities in the dark. Fire fighting UAVs may play a critical role at night by eliminating the danger to humans and using arrays of sensors to peer through the night. Boid algorithms would be critical for night time collision avoidance.

3.4.2 Automated Highway Systems

Automating commuter vehicles is a long-term goal for many agencies. Computers have faster reaction times than humans, and so can drive at higher speeds, and reduce vehicle collisions due to inattentive drivers. Putting computers in control also allows the vehicle occupants to better utilize the time they would have otherwise spent driving.

There are numerous complexities involved with automatic commuters, such as entry and exit from roads, interactions with other automatic commuters, interaction with human-operated vehicles, and reactions to emergency situations. Considering these challenges, using boid algorithms for guiding automatic commuters is a very attractive solution.

Boid algorithms require no inter-vehicle communication, as opposed to most other decentralized methods. The boids need only be able to sense one another. This allows boid vehicles to interact with other boid vehicles and human-operated vehicles alike.

Boid algorithms also have built-in collision avoidance methods, both for avoiding roadway obstacles and for avoiding other vehicles. In addition, entry and exit situations may be incorporated as behavioral rules. This further emphasizes the inherent flexibility of boid algorithms.

3.4.3 Battlefield Swarming

Using swarming tactics in military activities has a long history of effectiveness [10]. Swarming has been used for transit (such as Napoleon did when fighting Austria), for severing lines of communication, and for psychological effect.

The psychological effect of swarming tactics is found in the elusiveness and formlessness of

swarms. Enemy forces can become confused and disoriented by effective swarming. Swarms of UAVs can have a further psychological impact on ememy forces. The sight of a sky blackened by hordes of aircraft can be terrifying and perhaps break the resolve of the enemy combatants.

Swarms of UAVs can act as three-dimensional skirmishers to mask or protect higher-value aircraft. Using low cost shields of this nature could have a major effect on mission success rates. Such skirmishers could absorb ground or air fire, or even engage enemy forces without risking the protected aircraft.

In each of these battlefield cases, it is critical to use the lowest-cost aircraft system available. Using boid algorithms for guidance also reduces the number of personnel required for mission support. Boid algorithms, as discussed, have tremendous savings in on-board computational requirements as compared to other distributed guidance techniques.

CHAPTER 4. BOID RULES AS GUIDANCE

Of all the algorithms proposed for decentralized control, boid algorithms are by far the easiest to understand and implement. Boid algorithms have the added advantage of requiring very little on-line computation during simulation and flight, allowing more aircraft to be simulated at one time and a greater range of simulated missions. Yet these algorithms can still yield very complex behaviors. This chapter will further explain the ideas behind boid algorithms and how they may be translated to aircraft. Information on approaches other than the one taken in this paper will also be given.

For boids, the only given command is acceleration. However, the commands issued from the boid algorithm can not be implemented without some modification. For instance, the acceleration must be limited such that the speed limits and the maximum load factor are not exceeded.

4.1 Behavior Weightings

For each behavior of the boid algorithm a unit acceleration is calculated. The accelerations for each of the behaviors are given a weighting and then added together to obtain the total dimensionless acceleration. These unit acceleration values are then multiplied by a scale factor for translation to the actual physical acceleration. Such weightings are notably key to a successful guidance result. By adjusting the weightings relative to each other even slightly, vastly different swarm behaviors can emerge.

The boid algorithm implementation employed here utilizes five behaviors. The three basic behaviors of flocking, velocity (heading) matching, and aircraft collision avoidance are combined with the seeking and obstacle avoidance behaviors.

The weighting for each behavior is specified as a percentage of the sum of all the weightings. Setting these weightings requires very careful consideration. Work by Crowther [7] has shown it

is possible to achieve specific behaviors by adjusting the ratios of particular weightings. However, Crowther utilizes a load factor command in addition to a velocity command. This implementation uses only position and velocity commands due to limitations of the autopilot software, so the results obtained by Crowther are not directly applicable. It is also possible to optimize the weightings as will be discussed in Chapter 5.

4.2 Contingency Planning

In the course of a typical mission, many instances occur in which the rule weightings need to be changed. For instance, if a boid were to drift within the safe radius of its nearest neighbor, most of the available control power would need to be given to the anti-collision behavior in order to prevent a collision.

The user defines five parameters used to trigger the contigency management actions. A "Safe Obstacle Distance" parameter specifies the closest allowable approach to any obstacle. Violation of this barrier triggers a "Level 1" flag for the obstacle avoidance behavior. If the aircraft velocity vector intersects any part of the obstacle (shown in Figure 4.1), a Level 2 flag is triggered. The determination of a Level 2 flag is found in Equations 4.1 – 4.6. Otherwise, the default is a Level 3 flag.

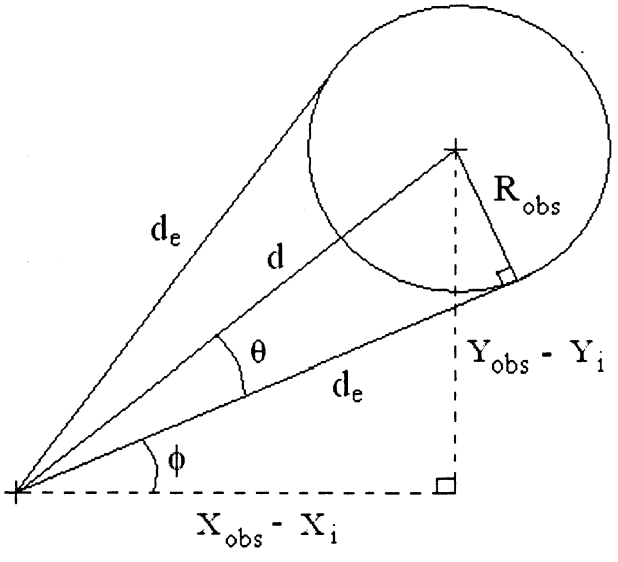

Figure 4.1 Obstacle edge finding schematic

$$d = \sqrt{(X_{obs} - X_i)^2 + (Y_{obs} - Y_i)^2} \tag{4.1}$$

$$\theta = \frac{\pi}{2} - cos^{-1}\left(\frac{R_{obs}}{d}\right) \tag{4.2}$$

$$\phi = cos^{-1}\left(\frac{X_{obs} - X_i}{d}\right) \tag{4.3}$$

$$d_e = d\ cos(\theta) \tag{4.4}$$

$$X_{edge1,2} = d_e cos(\phi \pm \theta) + X_i \tag{4.5}$$

$$Y_{edge1,2} = sgn(Y_{obs} - Y_i)\ d_e sin(\phi \pm \theta) + Y_i \tag{4.6}$$

Similarly, the "Safe Vehicle Distance" parameter triggers a Level 1 flag from the default Level 2 state for the aircraft collision avoidance behavior. The "Maximum Separation" value, if exceeded, triggers a Level 1 flag in the flocking behavior. The "Maximum Heading Difference" parameter triggers a Level 1 flag in the velocity matching behavior, if exceeded. Finally, the "Terminal Area Radius" parameter changes the flag for the seek behavior from Level 1 to Level 2 (the seek behavior default is Level 1 to simplify optimization procedures as will be discussed in a later section).

The flags for each behavior are combined into what will be defined as the "boid state". For each boid state, a particular set of behavior weightings is scheduled. As each boid progresses through its path, the boid state changes and the behavior weightings are adjusted. The weightings are set off-line either manually or through the use of optimization programs.

The settings for the contigency parameters are dependent partly upon the physical capabilities of the vehicle and partly on the desired buffer zone. For instance, upon violation of the Safe Obstacle Distance parameter, the aircraft cannot instantaneously turn to correct its path. Instead, there is a small amount of time spent within the safety zone during which the aircraft is adjusting its trajectory. Thus, there must be both a Safe Obstacle Distance which triggers the flag and an implied "Buffer Distance" which is the true minimum safe obstacle distance. The same is true for the other contigency parameters.

4.3 Applying Boid Rules to Aircraft

In order to use boid rules on aircraft, several modifications are required. First, the acceleration command generated by the algorithm must be translated into position and velocity commands. For this implementation, a standard ODE45 differential equation solver is used.

A very basic modification required of the boid algorithm for aircraft is speed limiting. Under the modifications, the program will not generate a velocity command above or below the respective maximum or minimum speeds, as specified by the user. These speed limits are usually defined by the stall speed and structural limitations of the aircraft.

A more complex addition to the boid algorithm, which will be discussed in detail in 4.3.2, is to limit the heading angle change of the boids. Limiting the heading angle change ensures the generated trajectories take into account the aircraft minimum turn radius.

Although this implementation does not consider it, an aircraft using a discrete-path system (as opposed to the waypoint-following system utilized herein) would need to address rate limiting. That is, the simulated aircraft must have a smooth transition between commanded speeds to reflect the actual physical limitations of the real aircraft. With waypoint-following systems, there is generally enough distance between the physical waypoints to "smooth over" any overly demanding velocity commands.

For clarity, a "waypoint" as defined here is the output from the boid algorithm. These waypoints are sent to the aircraft for use in its waypoint-following system.

4.3.1 Equations of Motion

The equations of motion used for the boid algorithm were derived from a simplified aircraft model. The model is shown in Equations 4.7 – 4.9. In these equations, $\dot{V}$ is the aircraft acceleration, T is thrust, D is drag, a_{boid} is the acceleration output of the boid algorithm, V_∞ is freestream velocity, ψ is heading angle, and $\dot{X}$ and $\dot{Y}$ are velocities in the X and Y directions, respectively.

$$\dot{V} = T - D = a_{boid} \tag{4.7}$$

$$\dot{X} = V_\infty \cos \psi \tag{4.8}$$

$$\dot{Y} = V_\infty \sin \psi \tag{4.9}$$

4.3.2 Turn Limiting

Taken as is, the boid algorithm can generate trajectories beyond the physical limits of an actual aircraft. To avoid such an occurence, turn limiting must be incorporated into the algorithm.

The turn limiting algorithm implemented in this research uses what will be defined here as the along-track (X_{track}) direction and the cross-track (Y_{track}) direction as seen in Figure 4.2. As shown, the algorithm defines the X_{track} direction as the direction of the line passing through the i-2^{th} and the i-1^{th} waypoints. If the magnitude (linear distance) of the i^{th} Y_{track} value is greater than the calculated maximum allowable value, the position and velocity values of the i^{th} waypoint are adjusted so that they fall within the range.

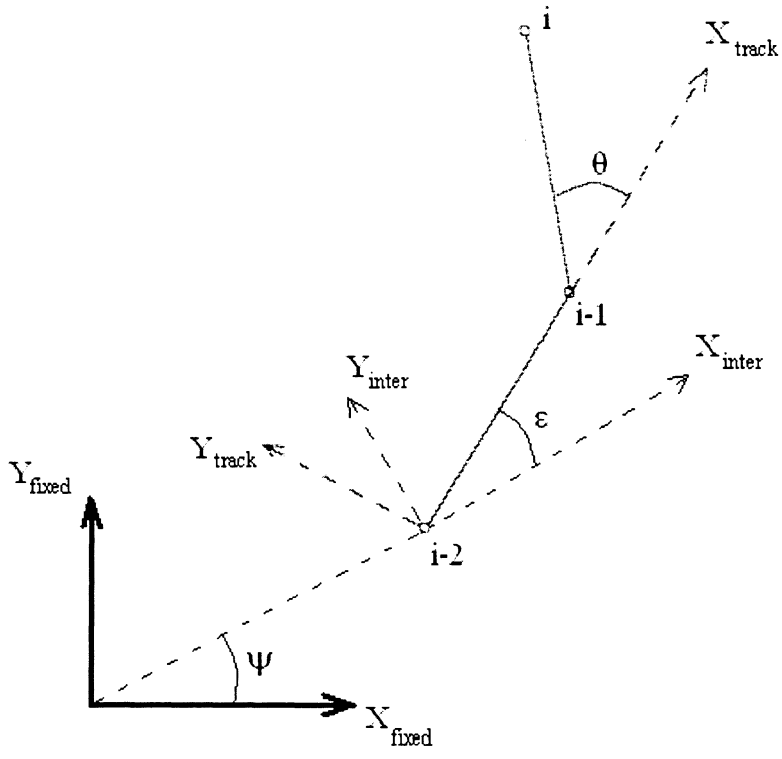

Figure 4.2 Schematic diagram for turn limiting

The minimum turning radius is calculated from the maximum load factor (n_{max}) shown in Equations 4.10 (constrained by the drag and maximum thrust) and 4.11 (representing the stall limit) [1]. In these equations, ρ_∞ is the freestream density, V_∞ is the freestream velocity, K is the

drag constant, W is the aircraft weight, S is the planform area, T is the thrust, and $C_{D,0}$ is the skin friction drag constant. Both equations are calculated for the particular freestream velocity and the lower value is chosen as the actual n_{max}. The minimum turn radius (R_{min}) is calculated according to Equation 4.12 [1].

$$n_{max} = \left\{ \frac{\frac{1}{2}\rho_\infty V_\infty^2}{K(W/S)} \left[\left(\frac{T}{W} \right)_{max} - \frac{1}{2}\rho_\infty V_\infty^2 \frac{C_{D,0}}{W/S} \right] \right\}^{\frac{1}{2}} \tag{4.10}$$

$$n_{max} = \frac{1}{2}\rho_\infty V_\infty^2 \frac{(C_L)_{max}}{W/S} \tag{4.11}$$

$$R_{min} = \frac{V_\infty^2}{g\sqrt{n_{max}^2 - 1}} \tag{4.12}$$

Since the design space is discretized, challenges arise in relating the minimum turning radius to the discrete points in time. In order to calculate the maximum allowable heading angle change, θ_{max}, it must be related to R_{min}. Approximating the sector angle for a circle of radius R_{min} and using trigonometric relations accomplishes this task. Assuming the sector angle, ϕ, remains small, the small angle approximation yields D $\approx$ S. The angle ϕ may thus be calculated as shown in Equation 4.13. If the time step is too large, however, the small angle assumption is invalid.

$$\phi = S/R_{min} \approx D/R_{min} \tag{4.13}$$

The position of the i_{th} point in the "track" coordinate system is determined using Equations 4.14 and 4.15. Using these coordinates, θ_{max} may be found simply by Equation 4.16.

$$X_i = R_{min} \cos\left(\frac{\pi}{2} - \phi\right) \tag{4.14}$$

$$Y_i = R_{min} \sin\left(\frac{\pi}{2} - \phi\right) \tag{4.15}$$

$$\theta_{max} = \tan^{-1}\left(\frac{Y_i - Y_{i-1}}{X_i - X_{i-1}}\right) \tag{4.16}$$

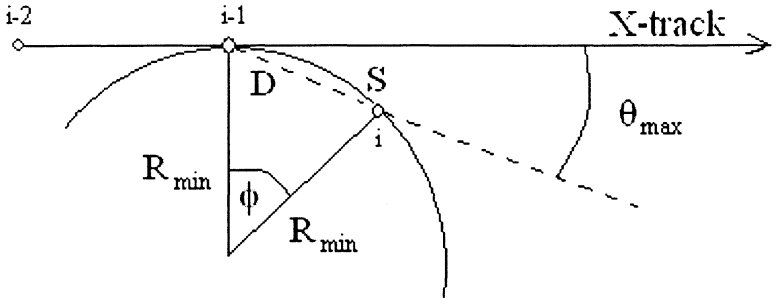

Figure 4.3 Calculation of θ_{max}

4.3.3 Global Position Mapping

In order to use the position values produced by the boid algorithm on an aircraft, they must be transformed into coordinates of latitude and longitude. Since the flight tests will be localized (covering less than 4 square miles) the conversion was simplified to ignore the curvature of the Earth. The simplified Equations 4.17 and 4.18 were used to calculate the latitude and longitude, respectively. Here, Ψ_0 and Ω_0 are initial latitude and longitude, and h_{scale} is the conversion from degrees to the particular distance unit used. For converting to nautical miles, the value of h_{scale} is $\frac{1}{60}$. It is assumed that the Y-direction is aligned with the longitude of the Earth and the X-direction with the latitude.

$$\Psi = \Psi_0 + h_{scale} Y_{pos} \tag{4.17}$$

$$\Omega = \Omega_0 + h_{scale} X_{pos} \tag{4.18}$$

4.3.4 Area Containment

It is important that aircraft do not violate the prescribed flight boundaries, particularly when flight testing. The boundaries are in place to minimize risk to the general public, facilities, and the flight testers themselves.

A buffer zone was established a specific distance within the boundaries to ensure the limits of the testing area are not violated. This buffer distance is defined as the amount required for an aircraft at full speed to completely reverse direction with maximum acceleration. Clearly, this distance is different for each type of aircraft. With increased confidence in the performance of the algorithms this strict buffer distance may be relaxed for increased mission flexibility.

The formulation of this activity (so-called because it is not truly a "behavior" as defined in this paper) is fairly straightforward. In simplest terms, if the buffer zone is violated then Equations 4.19 and 4.20 are called to command the new x and y accelerations. Here, a_{max} is the maximum acceleration for the vehicle and $F_{sign}^{(x,y)}$ is a function which yields ± 1. $F_{sign}^{(x,y)}$ takes as input the current position and the boundaries of the specific problem to determine whether a positive or negative acceleration (in the global coordinate system) is required to return the aircraft to the safety zone.

$$a_x = F_{sign}^{x}(x,y)\ a_{max} \tag{4.19}$$

$$a_y = F_{sign}^{y}(x,y)\ a_{max} \tag{4.20}$$

The concern with this approach is that the boids give full priority to area containment at the expense of collision avoidance behaviors. Since the safety of the public and operators is paramount for flight tests, however, it is a secondary concern.

4.4 Enhanced Boid Algorithms

4.4.1 Boids in Higher Degrees of Freedom

The equations given in this chapter are for an arbitrary number of spatial dimensions. In this implementation, the trajectories were calculated in two dimensions. Vertical collision avoidance

is mitigated in flight tests by altitude separation. Several reasons exist for not including higher-order models. The fact that a third dimension is not necessary to validate the boid algorithm and the non-triviality of coordinated vertical aircraft movement are two such reasons.

Including higher order coordination introduces several challenges. Flight in the vertical dimension requires consideration of an additional rotational axis, namely flight path angle. The resultant 6-DoF model is highly coupled and highly non-linear. The full aircraft equations of motion are shown in Equations 4.21 – 4.26, where V is the aircraft speed, g is gravitational acceleration, T is thrust, D is drag, W is gross weight, γ is flight path angle, n is the load factor, ϕ is the bank angle, and χ is the heading angle [12].

$$\dot{V} = g\left[(T-D)/W - \sin\gamma\right] \tag{4.21}$$

$$\dot{\gamma} = (g/V)\left[n\cos\phi - \cos\gamma\right] \tag{4.22}$$

$$\dot{\chi} = (gn\sin\phi)/(V\cos\gamma) \tag{4.23}$$

$$\dot{x} = V\cos\gamma\cos\chi \tag{4.24}$$

$$\dot{y} = V\cos\gamma\sin\chi \tag{4.25}$$

$$\dot{z} = -V\sin\gamma \tag{4.26}$$

Clearly, there is a great deal of work involved in coordinating motion in a 6-DoF environment. Research has been performed on some aspects of this problem, however. One technique for coordinating boid movements in attitude can be found in Bauso, et. al. [2].

4.4.2 Exponential Scaling Model

The most widely accepted practice for scheduling boid behavior weightings is by using exponential scaling. For instance, as a boid approaches an obstacle, the obstacle avoidance behavior weighting is increased exponentially. As a boid grows farther away from its neighbor, the flocking behavior weighting is increased. This model has seen a great deal of success in simulations.

Exponential scaling of the behavior weightings was not chosen for this implementation as one of the goals for this research was to minimize the amount of necessary on-line computing power required. While this model does not require unreasonable amounts of on-line computation, it does

require more than the basic model. Guiding this paper is the belief that a viable flight system may be produced using only the basic model with optimized contingency management, and so exponential scaling was not used. Additionally, for a truly flightworthy system, the exponential scaling model would still require a contingency management system. This approach would greatly complicate the weighting determination.

A fully realizable flight system using a contingency management alone may not be possible. In that case, the weighting system will need to include exponential scaling of weightings in addition to the contingency management system.

CHAPTER 5. OPTIMIZATION OF BOID ALGORITHMS

The application of the boid algorithm to aircraft was explained in detail in the previous chapter, but the very important subject of selecting the weightings was left open. Performing a study of the weighting ratios such as explored by Crowther [7] is a very complex and worthwhile task, although beyond the scope of this research. Optimization therefore remains the best option to set the weightings.

Optimization of boid-based algorithms is difficult, however, especially when contingency planning is applied. A major challenge to quickly optimizing the behavior weightings is that the function to be optimized is very complex. One needs to run the entire simulation for each iteration of the optimization procedure, which can amount to very long convergence times. In addition, the function is highly coupled, has numerous discontinuities, contains numerical integrations, and uses logic loops. Clearly, optimizing this function is not a trivial task.

5.1 Classical Optimization

Many techniques of classical optimization are known, falling into three main categories. These categories are enumeration techniques, random walk techniques, and calculus-based methods. The most rudimentary type of optimization is simple enumeration. Enumeration consists of determining the cost for every point in a finite design space (or a discretized continuous space) and finding the solution with the lowest cost. For detailed problems, it is easy to see how computing an enumeration solution can quickly become infeasible.

Random walk methods are very similar to enumeration, except random points in the design space are selected and evaluated, rather than the entire set being evaluated. Although at first glance it may seem to be more efficient, a great deal of accuracy is lost.

Calculus-based methods use gradient information to find extremeties of the objective func-

tion. These methods are the basis for most work in optimization theory. Following from Lewis and Syrmos [18], calculus-based methods consist of calculating derivatives of the Hamiltonian, a function combining the objective function (or performance index) and the constraints to the problem. These methods do not tend to work well (if at all) on large, complicated functions or functions with discontinuities. In the case of boids there are numerous discontinuities and other complexities, making the use of calculus-based methods impossible.

5.2 Genetic Algorithms

Genetic Algorithms (GAs) are members of the same family of biologically inspired (although not necessarily bio-memetic) tools as boid algorithms and neural networks. Proposed by John Holland in 1975 [15], GAs were created to better understand adaptation processes and how these processes could be applied to artificial systems. Genetic algorithms use an evolutionary system to produce emergent behavior.

Although GAs have found a niche within the field of optimization, they are not designed to be optimizers. The basic function for GAs is to explore and adapt to complex and time-varying fitness landscapes [8].

GAs are based on a "survival of the fittest" strategy. From an initial population set of binary strings, the objective function is evaluated and the strings with the best results are retained. The GA then applies reproduction, crossover, and mutation to yield a new population of strings for evaluation. This process is continued until convergence or until the maximum number of generations is reached.

Genetic algorithms are different from classical optimization techniques for a number of reasons. The four main ones from Goldberg [11] are listed below:

1. GAs work with a coding of the parameter set, not the parameters themselves.

2. GAs search from a population of points, not a single point.

3. GAs use payoff (objective function) information, not derivatives or other auxialliary knowledge.

4. GAs use probabilistic transition rules, not deterministic ones.

Rather than obtain a "best" solution, genetic algorithms yield a "best-ever" solution. This approach represents a major departure from traditional optimization concepts which endeavor to find and prove the optimum solution.

5.2.1 Formulation

In biological terms, chromosomes contain the raw genetic information of life. In computation, chromosomes are represented by strings of numerical information. In most applications, each individual member of a population is represented by a binary string. It is possible to use numbering systems other than base-2, but binary offers many advantages.

All the individuals of a particular generation are concatenated into one chromosome. This binary chromosome is what is directly acted upon by the selection, crossover, and mutation rules.

The phenotype is the actual organism formed after interpreting the chromosome. The same concept applies when working with artificial systems. The chromosome is translated into a phenotype using a mapping function. The phenotype contains the information to be directly input into the evaluation function. In this case, the boid algorithm itself is the evaluation function.

5.3 Optimization Using Genetic Algorithms

Genetic Algorithms can be very useful as function optimizers. They are extremely robust for all kinds of functions because they can accomodate non-linearities or discontinuities of the function.

Due to the random nature of genetic algorithms, it is difficult to determine a convergence parameter. The solution can exhibit large fluctuations in objective function value, even if there is a general trend toward higher or lower values. This problem is typically mitigated by simply limiting the number of generations and checking the results to see if further computation is required.

5.3.1 Objective and Fitness Functions

Objective and fitness functions are very closely related but differ in significant ways. Objective functions are problem-specific and are used to assign an absolute value of fitness to each individual.

Fitness functions use the objective function values to rank each individual relative to one another. For genetic algorithms, the results obtained from the fitness function are used to help determine the number of offspring each member of the population is allowed.

Determining the objective function is the first challenge. One obvious choice for the boids problem at hand might be to minimize the "Level 1" flags (the highest priority contingencies, such as imminent obstacle collision) over a consistent number of time steps. Other choices could include weighting the Level 1 flags for each behavior differently and minimizing the sum. Determining the appropriate objective function is sometimes the most difficult task for an optimization procedure.

5.3.2 Selection Rules

Selection is the method by which individuals are chosen as parents for producing offspring. Using the results of the fitness function, the probability of an individual being selected to reproduce is determined. This determination is made using one of two basic types of methods.

One of the most common methods of selection is known as roulette wheel selection. Individuals are given a slice of a "wheel" based on their relative rankings. The wheel is then given a random virtual spin and the winning individual indicated by the pointer is thereby selected. The process is repeated until the population is filled.

Another common method is known as stochastic universal sampling. This method is constructed in a manner similar to that of the basic roulette wheel method except that rather than a single pointer, N evenly spaced pointers are utilized, with N being the population size. This method is faster than basic roulette wheel selection.

5.3.3 Crossover

Crossover is the actual process of individual reproduction. It is the transfer of parents' genetic material to produce new offspring which demonstrate traits of both progenitors.

Crossover can be of several types. The simplest operation is single-point crossover. Here, a certain location in the genome is chosen as the crossover point (for example, the fifth bit position). All the genetic information before or after that point is swapped between the two parents.

In multipoint crossover, up to $l - 1$ random crossover points are selected (where l is the

chromosome length). Genetic material is traded between the first crossover point and the second one, between the third and the fourth, and so on until the end of the chromosome is reached. This method yields a more comprehensive and robust exploration of the search space.

Uniform crossover uses a binary mask string of a length equal to that of the chromosome. For each bit position, a "0" indicates that the first offspring will use material from the second parent. A "1" indicates the first offspring will use material from the first parent. The second offspring uses the inverse of these rules. Uniform crossover is believed to prevent bias due to the length of the chromosome.

5.3.4 Mutation

Mutation is seen as the least critical of the three operators. Mutation is used to ensure a healthy infusion of new genetic materials such that the population does not lose diversity. Because of its low priority, mutation is used intermittently in the course of algorithm execution.

Mutation enacts a change on a single bit with a given probability, flipping a bit from a "1" to a "0" or vice versa. Mutation probabilities are usually on the order of 0.01 or 0.001 mutations per bit position per generation.

Mutation ensures there is a finite probability that any given part of the search space can be reached. It adds a robustness against becoming trapped in local minima, thus providing a good chance of finding the global minimum.

5.3.5 Reinsertion

The reproduction processes will not always yield the required full number of new individuals. When fewer than the required number of new individuals are produced, it is known as a generation gap. The generation gap may be specified as a parameter indicating the number of new individuals desired for each generation. When the generation gap is large, the propogation strategy is termed elitist. Small generation gap strategies are said to be steady-state or incremental.

5.3.6 Sample Execution

In an example process, four binary strings are defined as in Table 5.1. Applying the phenotypical mapping, these chromosomes yield the values seen in the third column of Table 5.1. In this case, the conversion from binary to decimal is the phenotypical mapping. The "black box" objective function happens to yield the values in the fourth column. Therefore string 1 is 30% of the total, string 2 is 10% and so on. For single-pointer roulette wheel methods, these percentages translate directly to the probabilities of each one being selected.

Table 5.1 Parameter values for sample GA execution

Number	String	Phenotype	Objective Value	% total
1	11111	32	15	30
2	00111	8	5	10
3	00011	4	10	20
4	00000	0	20	40

Using the basic roulette wheel method as an example, the "wheel" is spun four times, one for each progenitor. As an example, these spins happen to yield String 4 twice, String 1 once, and String 3 once. These four strings will interact to produce offspring.

The strings are paired at random, example pairings being one String 4 with String 1, and the other String 4 with String 3. If the crossover point is randomly selected as the fourth bit position, the crossover operations on the two pairs will take place as shown in Table 5.2.

Table 5.2 Sample crossover execution

Parent strings	Offspring strings
0 0 0 ‖ 0 0 1 1 1 ‖ 1 1	0 0 0 1 1 1 1 1 0 0
0 0 0 ‖ 0 0 0 0 0 ‖ 1 1	0 0 0 1 1 0 0 0 0 0

For this simulation, the mutation operator happens to mutate the last bit in Offspring 1. The new Offspring 1 thus becomes 00010. In most generations, mutation will not occur due to the very small mutation rate.

In this example, the reproduction and crossover operations happened to yield two offspring identical to their progenitor pair. Thus, a copy of String 4 and a copy of String 3 passed through

the reproduction operations unchanged. This same result would be obtained if the generation gap parameter were set to 0.5. In that case, the pool of available parents would be reduced by 50%, and only the first pair of strings would produce new offspring.

5.3.7 BackStep Method

One method for improving the optimization results is theorized here and labeled as the BackStep Method. Using this technique, after some specified generation interval, G_i, the algorithm will collect the top N_{IND} individuals (where N_{IND} is the population size) of all past generations. The new super-chromosome of these elite individuals is then reinserted as the new population. This method yielded promising results as will be seen.

There is a risk to using the BackStep method, however. Depending on the algorithm for finding the minimum value of the set, if there are multiple objective function results with identical values the method could favor strings either at the beginning or end of the set. This could result in a loss of diversity in the genetic material.

This BackStep Method, though simple, was not found in any of the literature. Therefore, its effectiveness must be carefully investigated before applying it to the problem at hand.

CHAPTER 6. SIMULATION ENVIRONMENT

The simulation tools used in this research were instrumental in the modeling and testing of the algorithms. This chapter is designed to help in accurately representing the results of this research by clearly identifying the simulation tools.

The boid algorithm used in this research was developed using Matlab, as was the Genetic Algorithm. Matlab was chosen due to its extensive simulation support and easy-to-use interfaces.

This research includes the use of flight hardware for testing the boid algorithm. As mentioned in Chapter 1, the flight hardware selected is known as the Piccolo autopilot system. This system has been used for hundreds of small UAVs both in simulation and in flight. The complete system includes an avionics box, a ground station, an operator interface program, and an aircraft simulator.

6.1 Matlab Boids Simulation

The boid algorithm was coded in the Matlab programming language. Matlab was also utilized to simulate the algorithm on a computer with a 3 GHz Pentium 4 processor and 512 MB of RAM.

In order to make the simulation and development of the algorithm more efficient, a GUI was created using Matlab's GUIDE design tool. A screen shot of the GUI is shown in Figure 6.1.

In order to integrate the boid algorithm with the Piccolo system, an interface developed by NASA Ames Research Center (ARC) was used. The Matlab boid code was converted to the C programming language using a commercially available Matlab-to-C compiler.

To assist in manually adjusting the numerous weightings, a Matlab graphical-user-interface (GUI) was created. A screen shot of this GUI is shown in Figure 6.2.

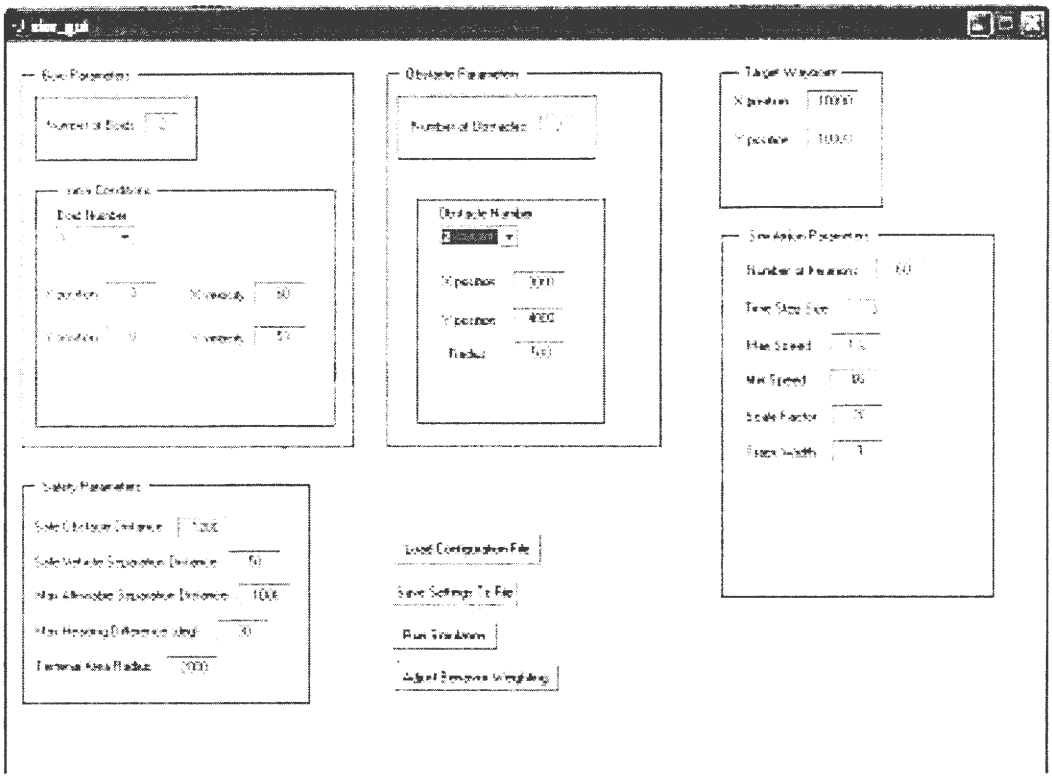

Figure 6.1 Boids simulation GUI

6.1.1 Genetic Algorithm Toolbox

This research used the Genetic Toolbox developed by Andrew Chipperfield, et. al. at the University of Sheffield to optimize the boid algorithm [6]. This freeware toolbox includes the basic methods of selection, crossover, and mutation in addition to even more advanced methods.

6.1.2 Intelligent Waypoint Algorithm

The Piccolo has a limited on-board memory storage capacity. The autopilot can retain only 99 waypoints at any given point in time. Paths that require more than 99 waypoints must be sent to the autopilot in stages, which adds more workload to the operator. It is therefore desireable for trajectories to have as few waypoints as possible. For instance, changing a long straight path section into just two waypoints at each end results in fewer intermediate waypoints. To this end, a simple "intelligent waypoint" algorithm was created. Using a technique similar to the turn limiting code of Chapter 4, X_{track} and Y_{track} directions are calculated. For convenience, Figure

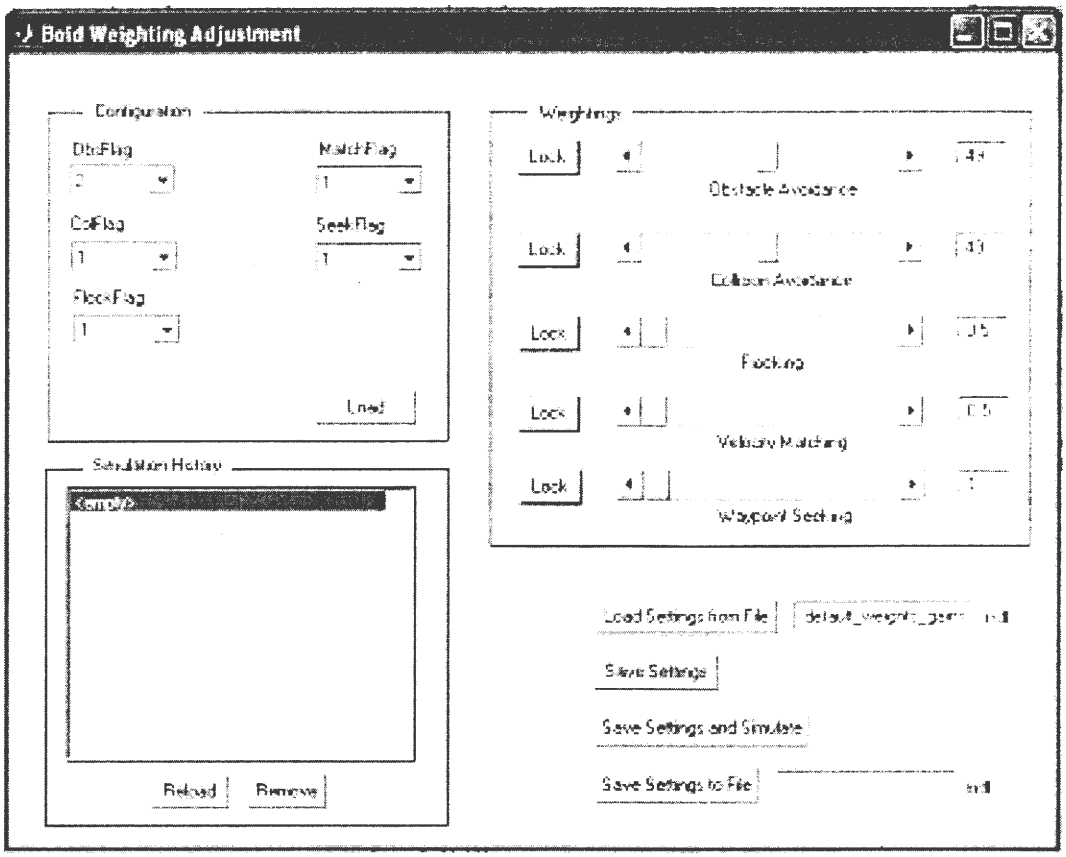

Figure 6.2 Weighting adjustment GUI

4.2 is reprinted here as Figure 6.3.

In the intelligent waypoint algorithm, the user specifies a "track width". Waypoints with a Y_{track} magnitude that is less than half of this track width are removed from the list. If a waypoint is removed, the X_{track} and Y_{track} directions are not updated, and the next waypoint is tested to see if it also falls within the range. This action is repeated until a waypoint falls outside of the range, indicating a less-than-negligible turn. Over the entire path, this procedure amounts to few waypoints on long, straight segments, and many waypoints on curvilinear segments. A sample application of the Intelligent Waypoint Algorithm is shown in Figure 6.4, where the original path is represented by a dashed line.

For the Intelligent Waypoint Algorithm to work properly, the user must carefully select the track width parameter. If the track width is chosen too large, large path deviations may occur in flight. The track width is application specific, and the user must thoroughly examine many

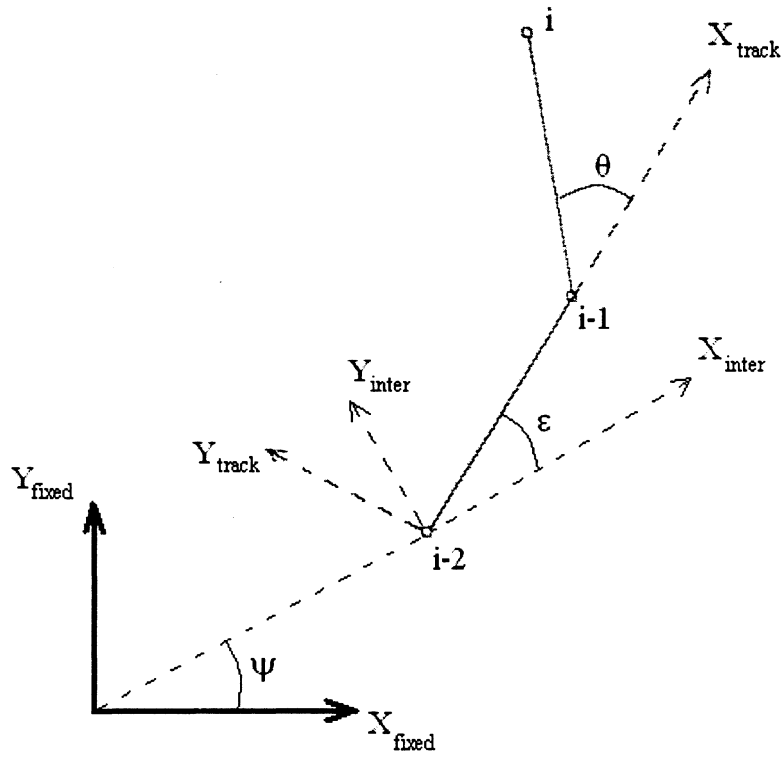

Figure 6.3 Schematic diagram for turn limiting

simulation runs to understand the effects of changes in this parameter.

6.2 Hardware-in-the-Loop Simulation

The Piccolo system provides for a real-time Hardware-in-the-Loop (HIL) simulation. The HIL simulation allows the designer to test algorithms and flight plans using the actual Piccolo autopilot avionics to be installed on the aircraft. This utility greatly reduces the chances of flight failure and saves development time.

The Piccolo HIL simulation requires one computer to run the Operator Interface and display the simulated graphics, and a computer for each avionics unit. Each Piccolo avionics box connects to its respective simulation computer using a Controller Area Network (CAN) interface bus. Each avionics box is also connected to the ground station using attenuated British Naval Connector (BNC) coaxial cables, simulating the Ultra-High Frequency (UHF) datalink used in flight.

The operator interface computer connects directly to the ground station. The simulation computers and the computer running the Operator Interface are connected via a User Datagram

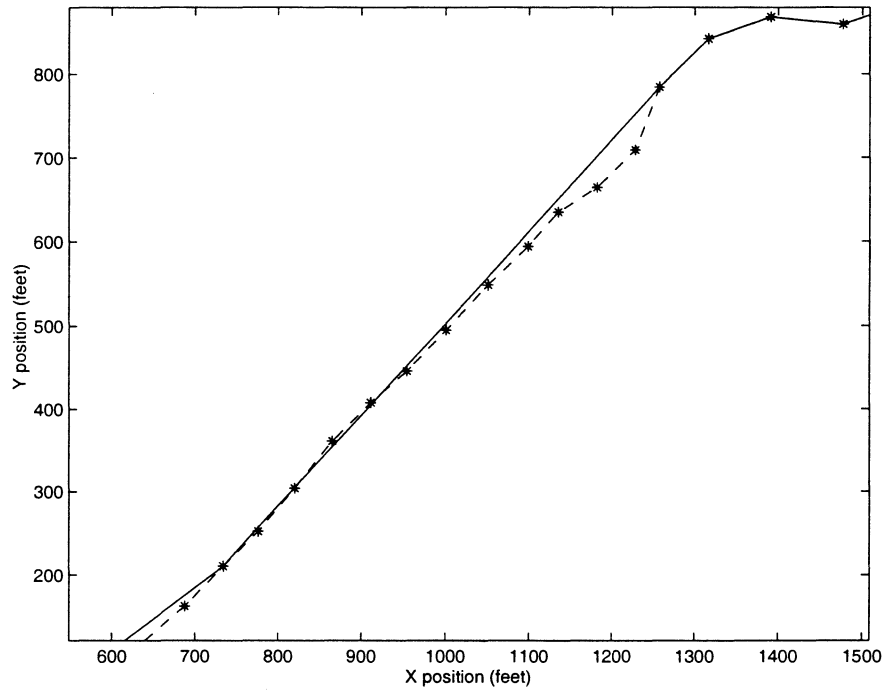

Figure 6.4 Sample application of the intelligent waypoint algorithm.

Protocol (UDP) network cable.

6.2.1 Software

6.2.1.1 Autopilot Control

The Piccolo software allows for control either manually (using a joystick or a standard R/C controller) or using the autopilot. Cloud Cap Technology (the manufacturer of the Piccolo) has performed extensive testing on the autopilot both in the lab and the field, giving this project confidence in the system reliability.

The autopilot control system was developed using Matlab's Simulink and autocoded to the C programming language using the Real-Time-Workshop and Stateflow. The physical plants and control laws were developed using this technique. The gains for the control laws may be specified by the user for the particular aircraft and the individual control loops may be disabled.

Automatic guidance is achieved in the Piccolo system using a waypoint-following technique. The user enters waypoint information consisting of desired latitude, longitude, and altitude. There is also the option to select an "Orbit" command for a waypoint, instructing the aircraft to orbit

the waypoint at a constant distance (this distance may not be specified in the current system, but the Piccolo II system will allow command of an orbit radius). There is no ability to command velocity with the default Piccolo system, but a speed controller developed by NASA ARC will allow this critical control element in the near future.

Using the interface provided by Cloud Cap, the waypoints must be entered manually. This is clearly unacceptable for an automatic guidance system as the initial conditions for the algorithm can not be known until flight. For this research paper, efforts will be made to consistently initialize the simulated flights. In the future and for the actual flight tests, a GUI developed by NASA ARC, which allows the automatic generation of waypoints during flight operations, will be utilized.

The Piccolo uses an on-line Kalman filter to estimate the aircraft attitude and the gyro biases. Global Positioning System (GPS) data is used to augment the filter. The use of such a system dramatically improves the robustness of the autopilot.

The winds aloft may be estimated by comparing airspeed against the GPS-based groundspeed. The autopilot can then account for the winds and remain on course, provided the winds do not exceed the capabilities of the aircraft. This provides important justification for using an open-loop system as explained in Section 6.2.1.3.

The autopilot is also capable of automatic landings and automatic catapult launches, although these feautures will not be used for this project.

6.2.1.2 Aircraft Simulator

The aircraft simulator provided by Cloud Cap allows for 6-DoF dynamics, wind models (including turbulence), aerodynamic models, and engine models. The aircraft models used are based upon the specifications of the APV-3 aircraft made by RnR Products in Milpitas, California. A photograph of this aircraft is shown in Figure 6.5 and the aircraft specifications are detailed in Table 6.1.

The aircraft model specifications are stored in look-up tables. The simulator uses linear interpolation to obtain any values that fall in between those stored in the tables. If a look-up table is not available, the simulator will estimate needed parameters in some cases. For instance, if the aerodynamics data table is unavailable, the $\frac{dC_L}{d\alpha}$ parameter will be estimated from physical

aircraft parameters and an assumed two-dimensional value of 2π.

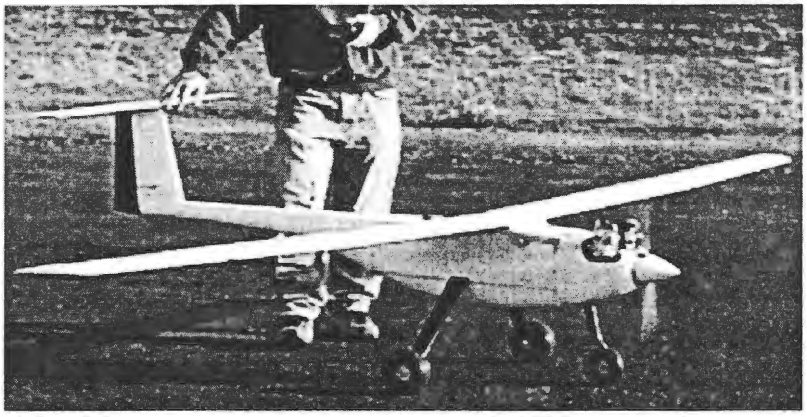

Figure 6.5 APV-3 aircraft made by RnR Products

Table 6.1 APV-3 aircraft specifications

Parameter	Value
Wing span	12 feet
Empty weight	30 pounds
Payload capacity	50 pounds
Maximum speed	90 mph
Minimum speed	45 mph
Maximum endurance	8 hours
Engine power	7.5 hp

6.2.1.3 Closed-Loop Guidance

Currently, the boid guidance system uses an open-loop method. This method is acceptable because of the excellent disturbance rejection of the Piccolo system. However, it prevents the management of extreme disturbances, pop-up obstacles, or mission modifications.

Since the boid algorithm can be computed extremely fast, it is possible to recalculate the trajectory at specified time-step intervals. This technique was not demonstrated as part of this work, but it is a fully realizable goal.

One reason for not using a closed-loop system in this implementation is that transmitting waypoint updates to the Piccolo avionics requires careful operator attention. This attention is

required because a change in the parameters of the current waypoint could yield unexpected and undesirable path changes. Thus, it is important that the insertion of new waypoints be performed very carefully. In addition, it was sometimes observed that when attempting to transmit new waypoints while an aircraft is flying, the paths would sometimes become garbled (incorrect waypoints would be assigned).

With these concerns in mind, and due to the fact that demonstration of closed-loop ability is not necessary to prove the effectiveness of the boid algorithms, it was determined that closed-loop guidance would not be investigated for this paper.

6.2.2 Hardware

6.2.2.1 Avionics

The avionics module of the Piccolo system (referred to only as "the Piccolo") is extremely compact and light, allowing installation on very light aircraft. A photograph of the module is shown in Figure 6.6. The module measures 4.8" x 2.4" x 1.5" and weighs 212 grams [28].

The Piccolo uses a 40 MHz MPC555 automotive microcontroller as its central processor. It is also instrumented with three rate gyros and two two-axis accelerometers, which allow for sensitive attitude and rate determinations. For measurement of air speed and pressure altitude, the Piccolo uses a dual-ported dynamic pressure sensor and an absolute barometric pressure sensor. Position and groundspeed information is determined using a differential-capable GPS antenna.

The Piccolo can support up to ten standard model aircraft servos. The Piccolo uses a radio modem datalink for command, control, and telemetry.

Using the information collected by the sensors, the Piccolo can estimate the airspeed, groundspeed, Euler angles, path information, vehicle health, and aerodynamic surface parameters. Clearly, the avionics module is very capable.

6.2.2.2 Ground Station

The ground station is used to link the computer running the operator interface to the avionics. The ground station can be used to control multiple Piccolo-based aircraft, both manually and

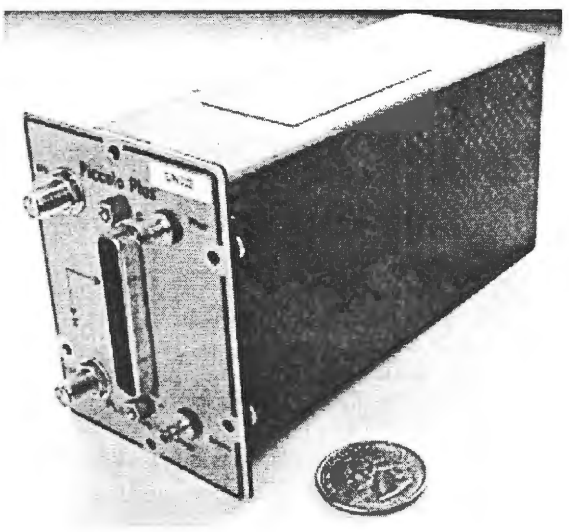

Figure 6.6 Piccolo avionics box made by Cloud Cap Technology

automatically. The datalink architecture allows multiple aircraft to be controlled by one ground station [28].

The ground station receives commands from the computer running the operator interface through a standard RS-232 connector. The ground station talks to the Piccolos using either the BNC cables or the UHF antennas.

The ground station allows manual control through a 6-pin Deitsche Industries Normen (DIN) connector. The operator selects which aircraft the pilot controls through the operator interface.

6.2.2.3 Computer Resources

The operator computer is a laptop with a 2.2 GHz Pentium 4 processor and 524 MB of RAM. This computer runs the operator interface program that connects with the ground station and displays the telemetry.

Each Piccolo requires a separate computer to run its aircraft simulation. The operator computer links to each simulator computer via the UDP connection. For simulating two aircraft, a laptop computer with a 2.66 GHz Pentium 4 processor and 256 MB of RAM and a desktop computer with an AMD Athlon processor and 524 MB of RAM were used.

All computers described in this section use the Microsoft Windows operation system. All computers use Windows 2000 except the aircraft simulator laptop, which uses Windows XP. The

use of Windows is required by the Cloud Cap software.

CHAPTER 7. SIMULATION RESULTS

7.1 Optimization Results

Preliminary optimization attempts focused on optimizing only one collection of weightings. In other words, the contingency flags could be triggered, but no subsequent change in the weightings would occur (i.e., no weight scheduling). This dramtically simplified the problem without losing much generality. Ideally, the optimized solution would be robust enough such that the contingency weightings could be set by hand using intuition.

7.1.1 Initial Optimization

The initial GA optimization sought to minimize the cost function $J = \Sigma(Obs_{L2} + Flock_{L1} + Match_{L1} + Seek_{L1})$, where *Obs* indicates the obstacle avoidance behavior, *Flock* indicates the flocking behavior, *Match* indicates the velocity (heading) matching behavior, *Seek* indicates the target seeking behavior, and the subscripts *L1* and *L2* represent Level 1 and Level 2 flags, respectively. (For information on how the flags are defined, please see Section 4.2.) In addition, a non-zero value of either the Level 1 obstacle avoidance flag or the Level 1 aircraft collision avoidance flag will trigger an arbitrarily high objective value of 8000. This trigger ensures that the final solution will not allow collisions of any kind.

The set of parameters used in the initial genetic algorithm optimization is shown in Table 7.1. A point of note is the allowable range for each behavior weighting is 0–100. However, with the boid algorithm implementation used, the sum of the behavior weightings must equal 100%. For simplicity, the weightings are determined by the ratios between the phenotype values. The resultant percentages are then used by the boid algorithm as the weighting values. This method is admittedly inefficient as it allows for multiple phenotypes to yield identical weightings.

The bit precision is set at 8, meaning there are 2^8 possible values for each of the five behavior

weighting variables. This implies a total of 2^{40} unique string possibilities, yielding a very large design space.

The generation gap was set at 0.9 which translates to using an elitist strategy for selection. With this strategy only the "best of the best" survive from generation to generation.

The set of simulation parameters used for testing the optimizer is found in Table 7.2. For clarity, heading is defined here according to a standard compass with 0° indicating up (North) and 90° indicating right (East).

Table 7.1 GA parameters for optimization

Parameter	**Value**
Number of Individuals	20
Generation Gap	0.9
Number of Generations	500
Number of Variables	5
Bit Precision for Variables	8
Variable Range	0–100

Table 7.2 Simulation parameters for optimization testing

Parameter	**Value**
Number of Boids	2
Boid 1 Initial Position	(100,0) feet
Boid 1 Initial Velocity	80 ft/s
Boid 1 Initial Heading	90°
Boid 2 Initial Position	(100,500) feet
Boid 2 Initial Velocity	80 ft/s
Boid 2 Initial Heading	90°
Number of Obstacles	1
Obstacle Position	(5000,5000) feet
Obstacle Radius	500 feet
Target Point	(10000,10000) feet

Using the cost function $J = \Sigma(Obs_{L2} + Flock_{L1} + Match_{L1} + Seek_{L1})$ produced results in which the boids would remain together but ignore the target. In fact, the GA would often produce results with W_{seek} equal to zero. To avoid this problem, the $Seek_{L1}$ parameter in the cost function was multiplied by a constant to elevate its importance above the other three parameters. This was justified by the fact that it is most important that the vehicles reach the target area (without collision), yet still important that they do so as a group.

7.1.2 BackStep Comparison

A simple comparison test of the BackStep method discussed in Chapter 5 was performed. Figure 7.1 shows the best objective function values for both the standard Simple Genetic Algorithm (SGA) and the BackStep method over 500 generations. The figures clearly show that the BackStep method reaches convergence faster than the SGA. Therefore, the BackStep method was used for most of the boid algorithm optimization in this paper.

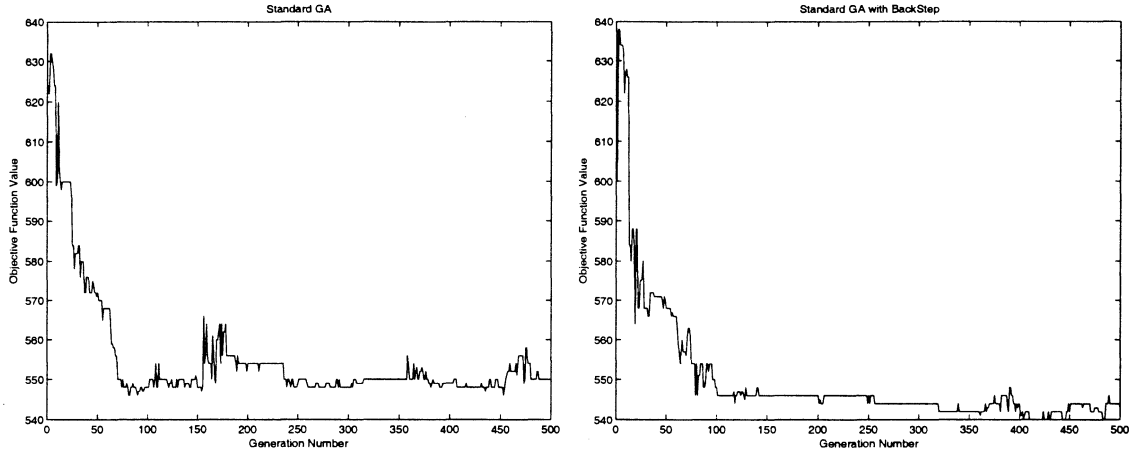

Figure 7.1 Comparison of 2 GA methods

7.2 Matlab Transit Simulation

Since a full optimization of the contingency planning was not possible, only critical parts of the contingency management were used for this simulation. The transit simulation in Matlab was configured such that it used the optimized paths for all boid states except for those where the safe obstacle or safe vehicle separation distances were violated. If vehicles drift too close together, the anti-collision weighting is given full authority. If a vehicle violates the obstacle safety distance, the obstacle avoidance weighting is given full authority. In the event both parameters are violated, the anti-collision weighting is given full authority, as it prevents the loss of both vehicles rather than just one.

Two boids maneuver between two obstacles in Figure 7.2 (note that the boids are traveling towards the 'X'). The minimum safe obstacle distances are shown encircling the obstacles. The

obstacles have radii of 500 feet and the safe obstacle distances are set at 100 feet.

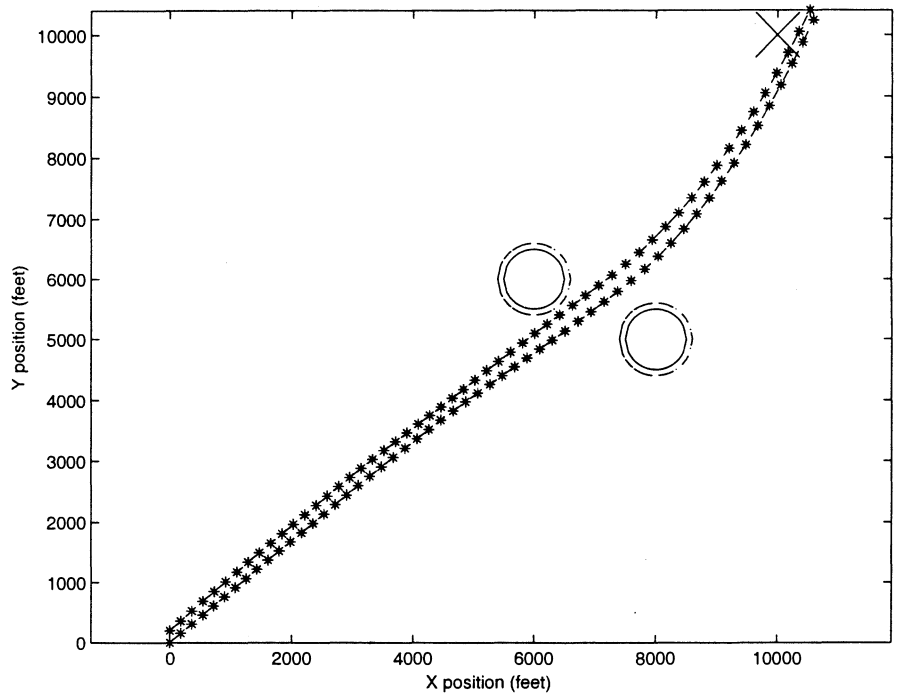

Figure 7.2 Two boids maneuvering between obstacles.

The future flight tests will take place at Edwards Air Force Base in California. A satellite image of the planned flight area is shown in Figure 7.3. In the image, the flight operations team will be located at the position marked by the star. The 'L' shape of the flight area is due to a number of considerations, the main one being avoiding overflight of the team. The area containment feature is therefore very important for these flights.

The finalized test plan will consist of four transits. The first is shown in Figure 7.4, the second is shown in Figure 7.5, the third is shown in Figure 7.6, and the final transit is shown in Figure 7.7. This back-and-forth flight plan maximizes the use of flight resources while gradually increasing the complexity of the mission.

The first flight (in Figure 7.4) shows a demonstration of the area containment feature. The 'L' shape of the flight area is representative of the actual containment area for flight tests. The buffer distance is set at 600 feet. The full simulation parameter set may be found in Table 7.3. The results of the third test, with the same configuration except for the addition of two 100-foot

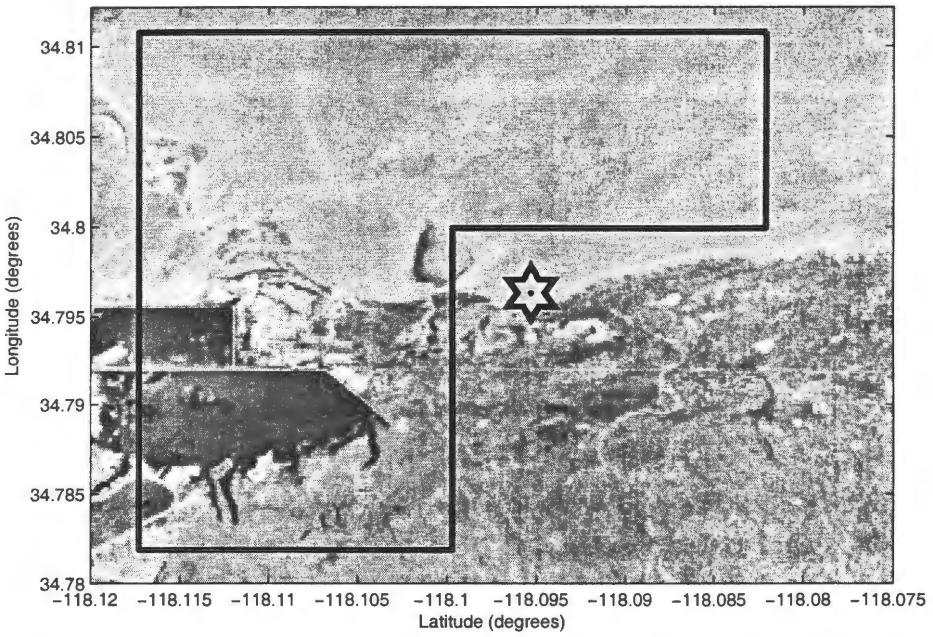

Figure 7.3 Satellite image of planned flight area

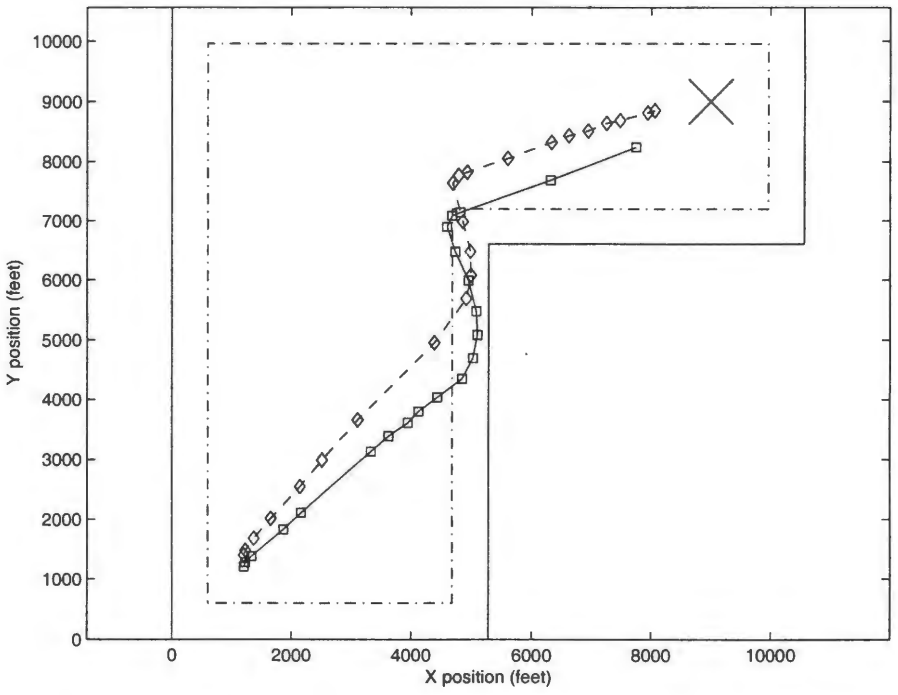

Figure 7.4 Boundary avoidance with two boids (first transit)

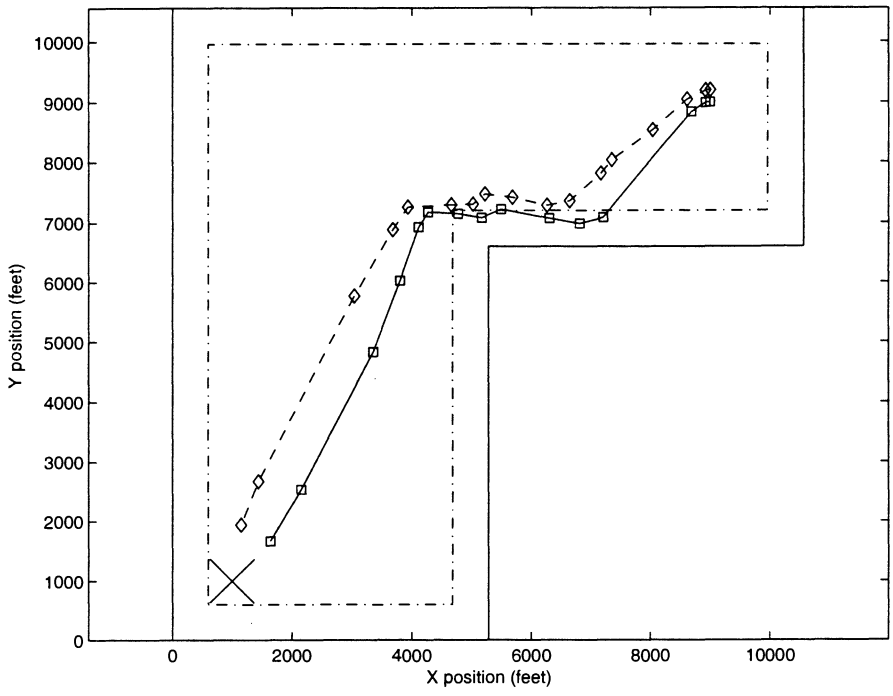

Figure 7.5 Boundary avoidance with two boids (second transit)

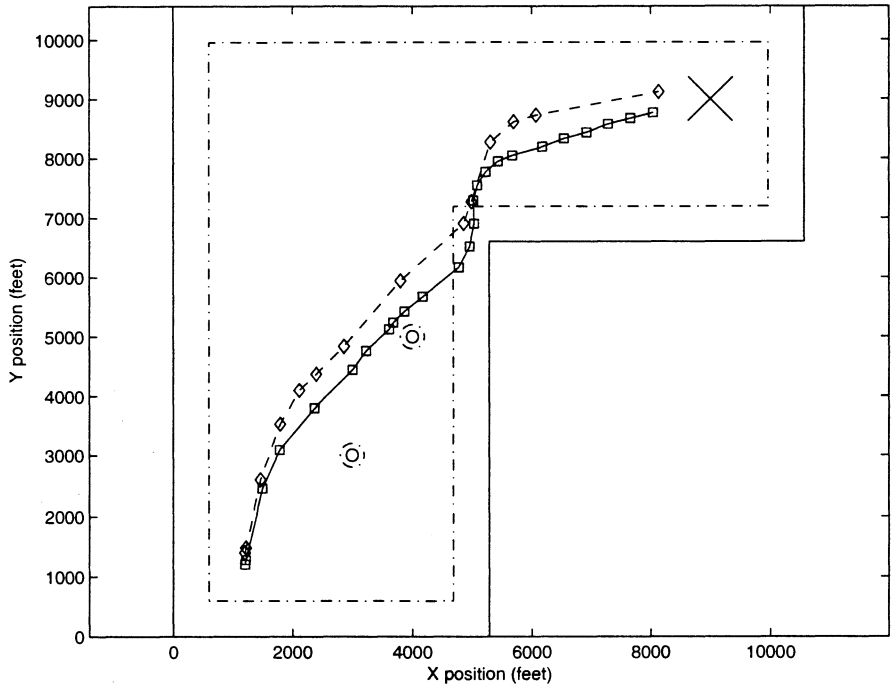

Figure 7.6 Boundary and obstacle avoidance with two boids (third transit)

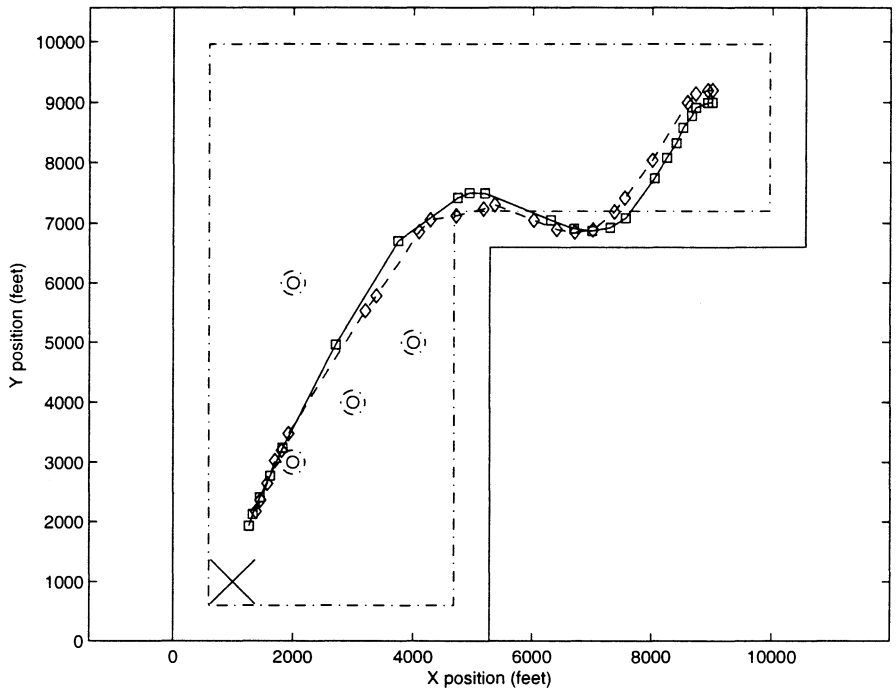

Figure 7.7 Transit simulation with four obstacles (fourth transit)

radius obstacles (with 100-foot safe obstacle distances), is shown in Figure 7.6.

The second and fourth flights have the same basic set of flight parameters as in Table 7.4. The fourth flight, again seen in Figure 7.7, includes four obstacles with locations and radii according to Table 7.5. All four obstacles have 100-foot safe obstacle distances.

Table 7.3 Simulation parameters for first and third flights

Parameter	**Value**
Number of Boids	2
Boid 1 Initial Position	(1200,1200) feet
Boid 1 Initial Velocity	80 ft/s
Boid 1 Initial Heading	20°
Boid 2 Initial Position	(1200,1400) feet
Boid 2 Initial Velocity	80 ft/s
Boid 2 Initial Heading	20°
Target Point	(9000,9000) feet

To test the robustness of the guidance system, several cases with different inital conditions were tested. In Figure 7.8, one boid was started with its velocity vector pointing opposite from the original setting in Table 7.4. As shown in Figure 7.9, the changed initial velocity has little

Table 7.4 Simulation parameters for second and fourth flights

Parameter	Value
Number of Boids	2
Boid 1 Initial Position	(9000,9000) feet
Boid 1 Initial Velocity	80 ft/s
Boid 1 Initial Heading	270°
Boid 2 Initial Position	(9000,9200) feet
Boid 2 Initial Velocity	80 ft/s
Boid 2 Initial Heading	270°
Target Point	(1000,1000) feet

Table 7.5 Obstacle specifications for fourth flight

Parameter	Value
Number of Obstacles	4
Obstacle 1 Position	(2000,3000) feet
Obstacle 1 Radius	100 feet
Obstacle 2 Position	(3000,4000) feet
Obstacle 2 Radius	100 feet
Obstacle 3 Position	(4000,5000) feet
Obstacle 3 Radius	100 feet
Obstacle 4 Position	(2000,6000) feet
Obstacle 4 Radius	100 feet

effect on the mission result.

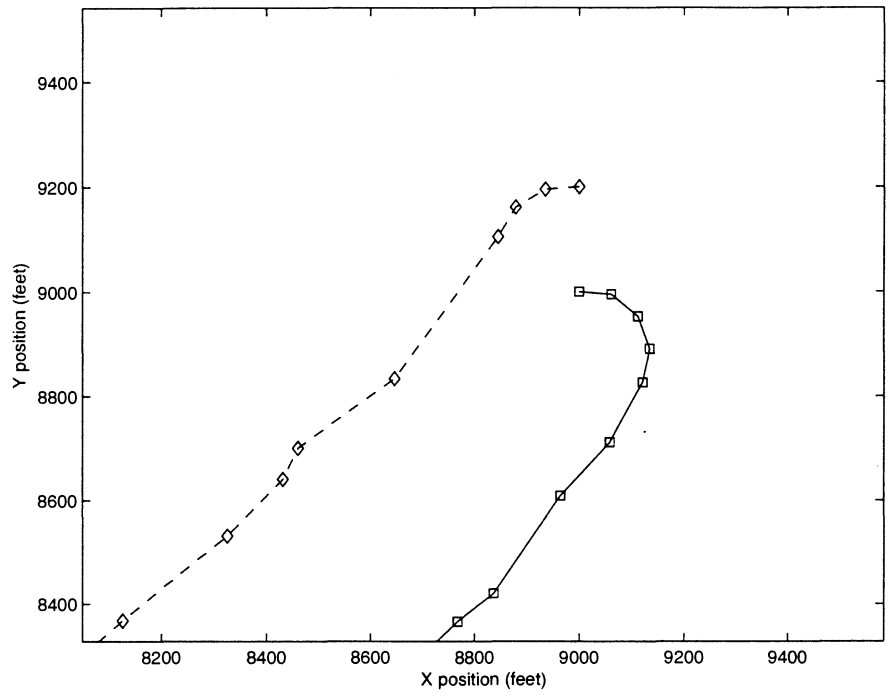

Figure 7.8 Magnified view of transit simulation with different initial velocity

7.3 Piccolo HIL Transit Simulation

The HIL tests are a cornerstone of this research. Successful HIL tests are essential to proving the flightworthiness of the boid algorithms. Figure 7.10 shows some sample results from the the HIL simulation with two aircraft utilizing boid guidance. The position tracks of the aircraft are laid over the commanded waypoints.

A section of the track is magnified in Figure 7.11. As shown, the aircraft track does not exactly follow the waypoint path. Due to the waypoint following logic of the Piccolo control system, the aircraft is not required to pass directly over the waypoints. It often uses the waypoints as a guide for performing turns instead of flying to them. This is an undesirable feature of the Piccolo software as it introduces marked uncertainty into the paths.

Tests with strong wind disturbances show that the waypoint following system is fairly robust. The results of a test with a simulated 40 mph wind from the left (West) are presented in Figure

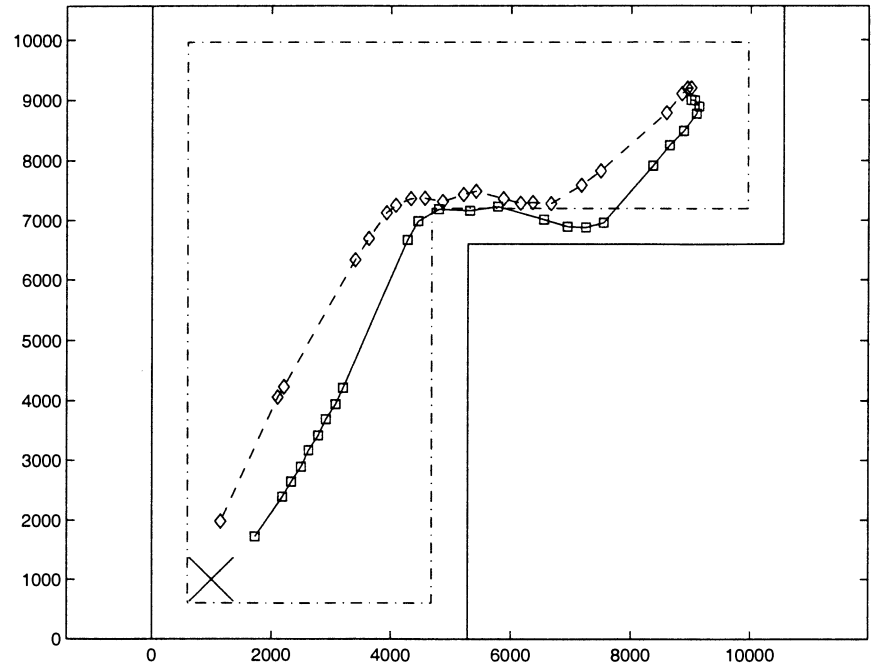

Figure 7.9 Expanded view of transit simulation with different initial velocity

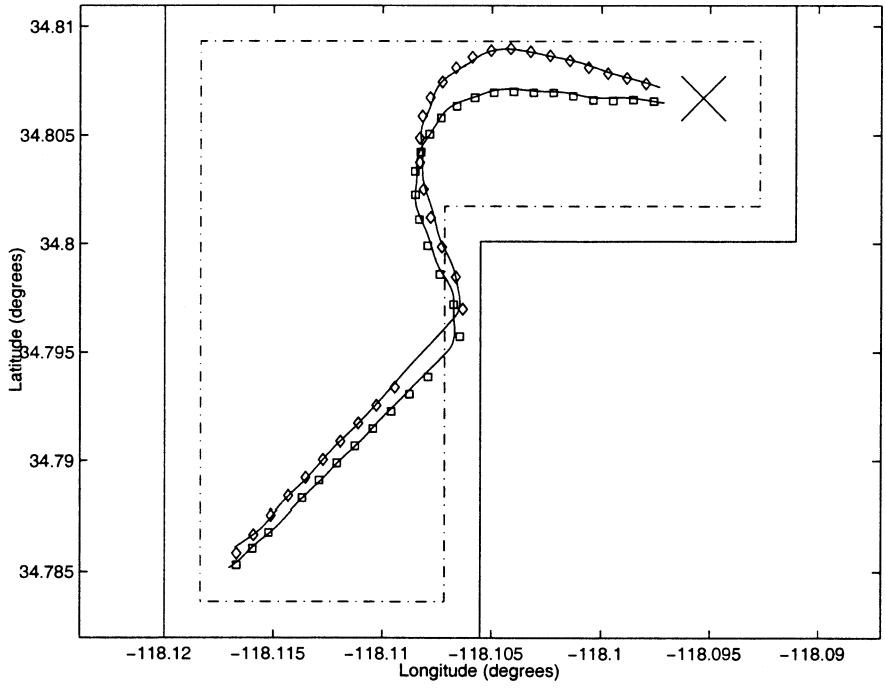

Figure 7.10 Sample results from HIL simulation

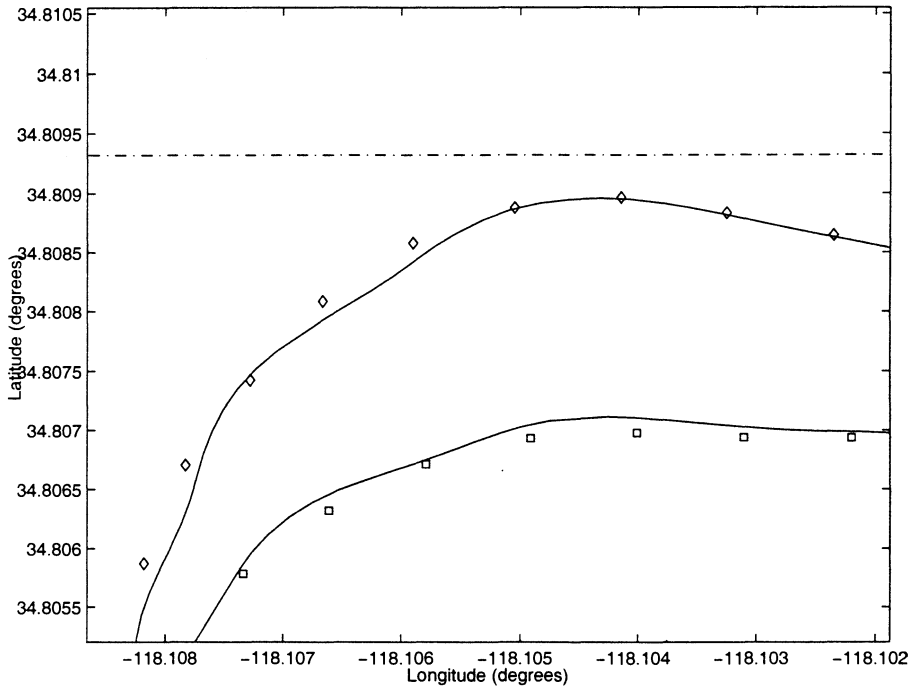

Figure 7.11 Magnified HIL simulation results

7.12. The resultant trajectories are clearly not as accurate, but for many applications this disturbance rejection should be adequate. The results of these wind tests thereby justify the use of an open-loop guidance system for this research.

7.4 Simulation of Many Boids

To further test the utility of the system, several simulations and optimizations were run with three or more aircraft. A sample run of five boids avoiding three obstacles is shown in Figure 7.13.

7.5 Matlab Orbit Simulation

A useful emergent behavior is the ability for a swarm to cooperatively orbit a common waypoint without collision. This behavior can be useful as a makeshift holding pattern or for creating a "mustering" area where aircraft wait after launch until all units may be deployed as a group.

The orbit behavior would be used after a transit when the aircraft have reached the terminal

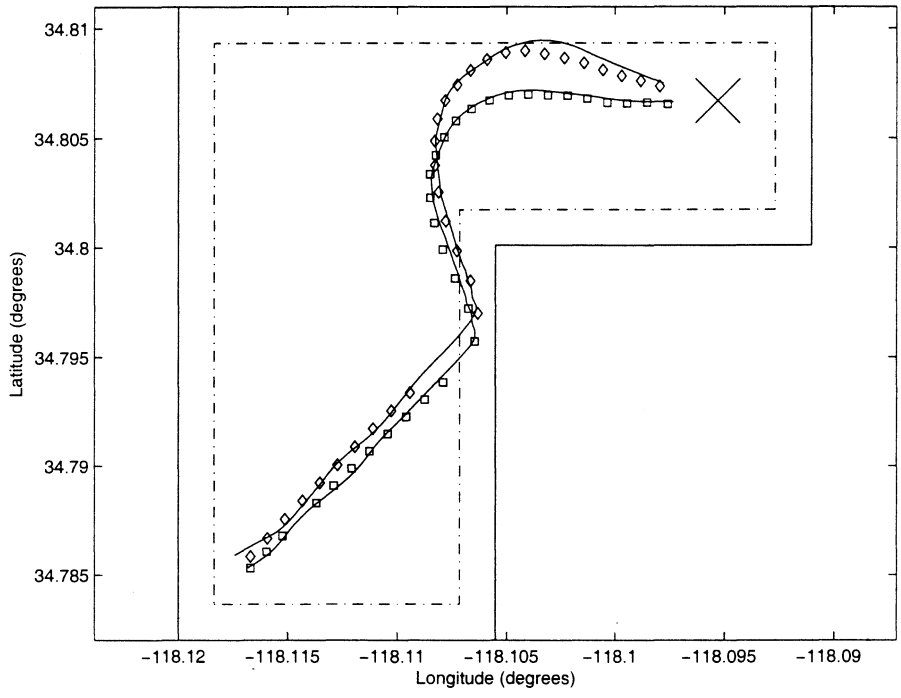

Figure 7.12 HIL simulation with wind disturbance

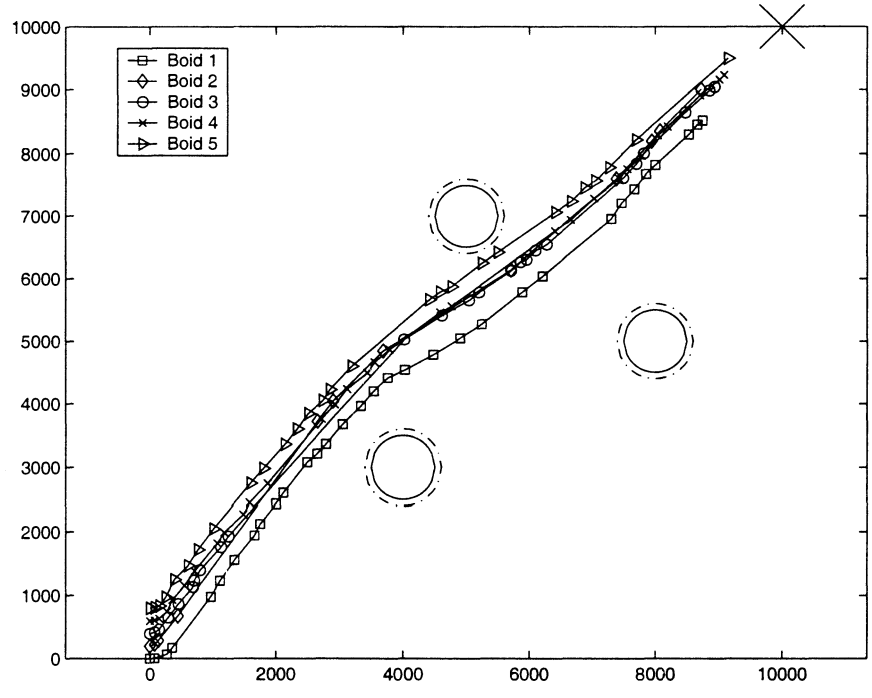

Figure 7.13 Simulation of 5 boids avoiding obstacles

area. The contingency management system would enable the "orbit" weights and the aircraft would orbit the desired target point.

A simulation was constructed such that the two boids were intialized with opposite velocities. The objective function for the optimizer was $J = \Sigma Seek_{L1}$, with a Level 1 flag in the anti-collision behaviors trigger $J = 8000$ as before.

Figure 7.14 shows the entry into the resultant emergent orbit behavior. Figure 7.15 is a magnified view of the orbit itself. The resultant orbit is very tight, although orbits with larger radii may be commanded by further limiting the minimum turn radius. Although difficult to see in the static picture, the boids are orbiting the target point opposite from one another. It is also interesting to note the slight precession of the orbits over time.

These results are very promising. They show that a particular emergent behavior can be commanded given an appropriate objective function. Interesting future work might investigate other potential emergent behaviors.

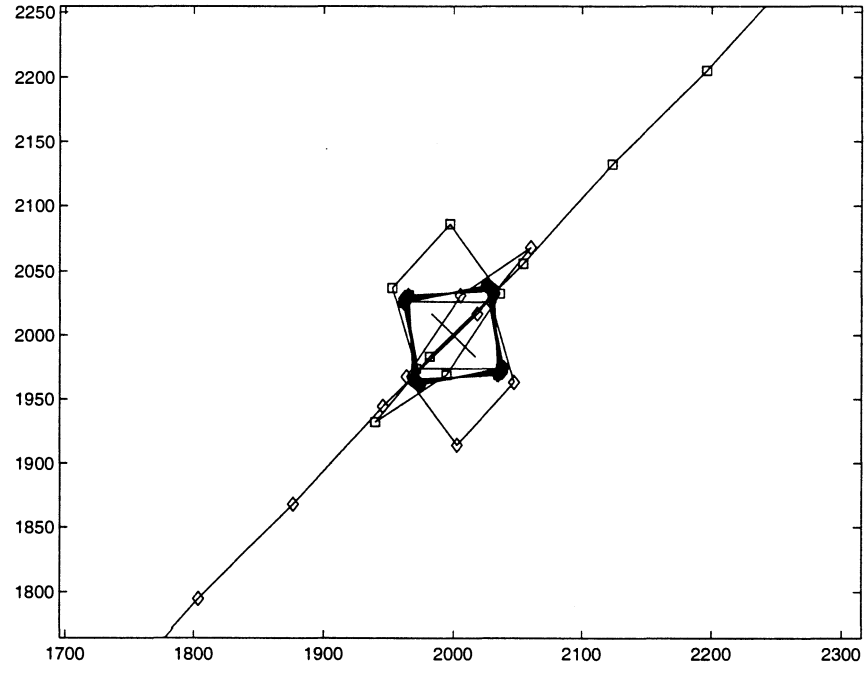

Figure 7.14 Simulation of emergent orbit behavior

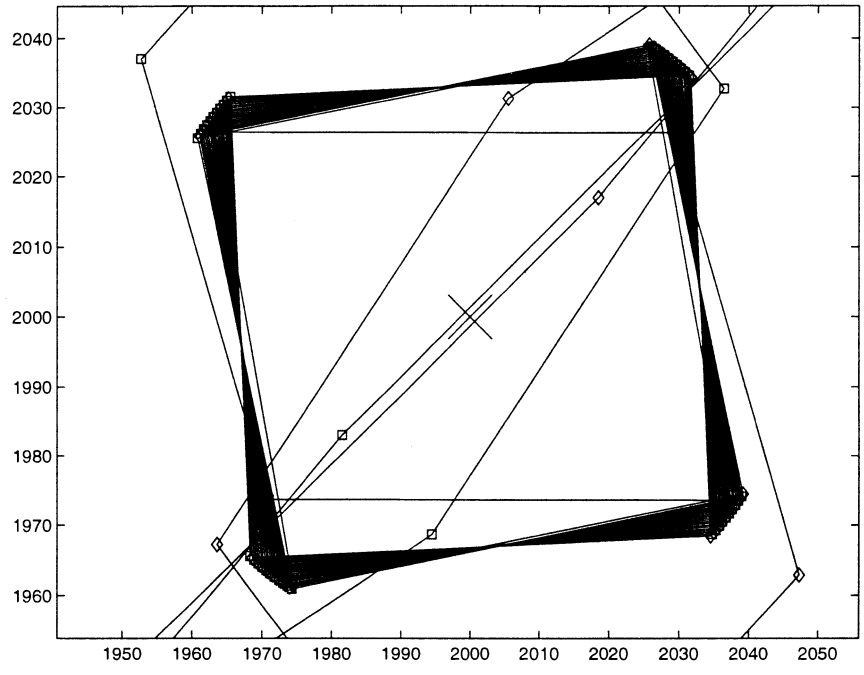

Figure 7.15 Magnified section of emergent orbit behavior

CHAPTER 8. DISCUSSION AND CONCLUSIONS

8.1 Discussion of Results

The simulations of boid algorithm guidance show that the system is both feasible and practical. Numerous simulations show the robustness of the boid algorithm with a very basic contingency management system. The full system, utilizing all optimized contingency parameters, would likely produce even more improved results.

As expected, the optimization of the boid behavior weightings did not produce a single set of values that can be applied to all situations without resulting in collisions. However, by optimizing a situation such that the safety zones are overly strict, a single set of weightings may be applied to many different mission configurations. In addition, simulation showed the magnitudes and directions of the initial velocities do not have a significant effect on the resulting paths. These results lend creedence to the belief that employing a fully optimized boid contingency management system will yield a universally applicable set of behavior weightings.

Simulations of more than two boids showed that the system is scalable. Increasing the number of boids does not significantly increase the required computation for each boid. Simulating the entire system, however, increases the overall computation time. This computation, of course, does not occur during flight.

This algorithm achieved the goal of low on-line computation requirements. In tests on a 3 GHz Pentium 4 processor with 512 MB of RAM, the computation of paths with about 100 waypoints each for two boids took about 1 second. The computation for each waypoint is then approximately 0.005 seconds. This on-line computation time indicates a computation requirement that is very low. These results are sufficient to declare success in this area.

This research shows that regardless of the technique used, guidance of multiple aircraft requires a large amount of computation. The difference between the technique used in this paper and

previous work is that here the computation is performed off-line. Off-line computation allows the flight hardware to focus on other tasks or may allow the reduction of the flight hardware itself.

Boid algorithms are conclusively proven herein to be suitable for use in guiding multiple aircraft. This important finding validates the need for further investigation of boid algorithms and their capabilities. Using this technology, fleets of fully autonomous and independent aircraft may soon become reality.

8.2 Lessons Learned

The primary lesson learned from this work was that very simple formulations can lead to extraordinarily complex results. Understanding the effect that adjusting parameters has on the emergent behaviors is extremely important.

Another important lesson learned was that many functions can not be optimized using traditional methods. Some problems are simply too large and complex for anything resembling classical optimization. In that regard, Genetic Algorithms represent a major advance in optimization research.

The Piccolo system uses a waypoint-following technique. Use of such a system means the path taken between waypoints is essentially arbitrary. Clearly, this system is not acceptable for robust collision avoidance. An operational system must use a true path-following technique, with appropriately small time steps between path points. However, a waypoint guidance system requires relatively little computation, making it a good choice for aircraft with limited computation power.

8.3 Future Work

An optimization of the full contingency management system is required for a complete guidance system. This requires a large amount of computation power, however, necessitating access to better computing resources. A full Monte Carlo analysis would also be required to gain complete confidence in the robustness of the boid algorithm.

Flight tests are critical to the full evaluation of this guidance system. Unfortunately, scheduling and budgeting constraints prevented the planned flight tests before the completion of this paper.

It is hoped that flight tests of the boid algorithms can be accomplished in the near future, further validating this work.

A prelimary investigation of specific emergent behaviors showed success in commanding an orbit about a target point. Further work could theorize and investigate commanding other emergent behaviors. This is a rich area for future research as the full range of possible behaviors is unknown.

Investigation of the thermal centering rule scheme proposed in Section 3.3.2 is an important area for future work. As discussed, using the energy advantage from riding thermals is a very desireable ability. Only preliminary research into automating thermal utilization exists and the true capabilities of such an aircraft (let alone fleets of such aircraft) can only be theorized.

The applications for boid rules found in Section 3.4 open many tantalizing possibilities. The concept of skirmishing UAVs using boid algorithms is very intriguing, and one that is very deserving of further study. Such technology could have a major effect on the future of air warfare. Automated commuter vehicles could revolutionize the way humans travel, making transportation more efficient and safe. The initial impetus for this paper, automated fire fighting support, may see the most immediate application of boid algorithms. Fire fighters may very shortly have the tools to wage safer and more effective campaigns against destructive fires with the help of UAVs under boid guidance.

REFERENCES

[1] Anderson, J. D. "Airplane performance: Accelerated flight" *Aircraft Performance and Design*, The McGraw-Hill Companies, Boston, 1999.

[2] Bauso, D., Giarre, L., and Pesenti, R. "Attitude alignment of a team of UAVs under decentralized information structure" *IEEE International Conference on Control Applications*, Istanbul, Turkey, June 2003.

[3] Bellingham, J., Tillerson, M., Alighanbari, M., and How, J. "Cooperative path planning for multiple UAVs in dynamic and uncertain environments" *IEEE Conference on Decision and Control*, Las Vegas, NV, December 2002.

[4] Blakelock, J. *Automatic Control of Aircraft and Missiles*, John Wiley and Sons, New York, 1965.

[5] Castillo, E., Conejo, A., Pedregal, P., Garcia, R. and Alguiacil, N. "Mixed-Integer Linear Programming" *Building and Solving mathematical Programming Models in Engineering and Science*, Wiley, New York, 2002.

[6] Chipperfield, A., Fleming, P., Pohlheim, H., and Fonseca, C. "Genetic Algorithm Toolbox for use with Matlab" *Department of Automatic Control and Systems Engineering, University of Sheffield*, Sheffield, England, 1995.

[7] Crowther, W. J. "Rule-based guidance for flight vehicle flocking" *I MECH E Part G Journal of Aerospace Engineering*, April 2004, *218*(2), 111–124.

[8] De Jong, K. "Genetic algorithms are not function optimizers" *Foundations of Genetic Algorithms (2)*, Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[9] Dimock, G. and Selig, M. "The aerodynamic benefits of self-organization in bird flocks" *41st Aerospace Sciences Meeting and Exhibit*, Reno, NV, January 2003.

[10] Edwards, J. "Swarming on the battlefield: Past, present, and future" *National Defense Research Institute (RAND)*, 2000.

[11] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[12] Giulietti, F., Pollini, L., and Innocenti, M. "Formation flight control: A behavioral approach" *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Montreal, Canada, August 2001.

[13] Hanson, C., Ryan, J., Allen, M., and Jacobson, S. "Flight test results for an autonomous formation flight control system" *AIAA 1st UAV Unmanned Aerospace Vehicles, Systems and Technologies Conference*, Portsmouth, VA, May 2002 (AIAA-2002-3431).

[14] Higdon, J. and Corrsin, S. "Induced drag of a bird flock" *The American Naturalist*, July 1978, *112(986)*, 727–744.

[15] Holland, J. *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, MI, 1975.

[16] How, J., King, E. and Kuwata, Y. "Flight demonstrations of cooperative control for UAV teams" *AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit*, Chicago, September 2004 (AIAA-2004-6490).

[17] Inalhan, G., Stipanovic, D., and Tomlin, C. "Decentralized optimization with application to multiple aircraft coordination" *4th International Conference on Cooperative Control and Optimization*, Destin, FL, November 2003.

[18] Lewis, F. and Syrmos, V. *Optimal Control*, John Wiley and Sons, New York, 1995.

[19] Mitchell, T. M. "Reinforcement Learning" *Machine Learning*, The McGraw-Hill Companies, Boston, 1997.

[20] Nash, J. "The bargaining problem" *Econometrica.*, 1950, *28*, 155–162.

[21] Park, C., Tahk, M., and Bang, H. "Multiple aerial vehicle formation using swarm intelligence" *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, TX, August 2003 (AIAA 2003-5729).

[22] Reichmann, H. *Cross-Country Soaring*, Soaring Society of America, Hobbs, N. M., 1993.

[23] Reynolds, C. "Flocks, herds, and schools: A distributed behavioral model" *Computer Graphics*, 1987, *21(4)*, 25–34.

[24] Richards, A., Bellingham, J., Tillerson, M., and How, J. "Coordination and control of multiple UAVs" *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Monterey, CA, August 2002.

[25] Robinson, D. "The Nash Solution for a bargaining game" *Laurentian University Department of Economics Website (www.economics.laurentian.ca)*, Ontario, Canada, 2000.

[26] Savant, C. "Elements of ballistic guidance" *Principles of Inertial Navigation*, McGraw-Hill, New York, 1961.

[27] Sigurd, K., and How, J. "UAV trajectory design using total field collision avoidance" *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Austin, TX, August 2003.

[28] Vaglienti, B., Hoag, R., and Niculescu, M. "Piccolo system user's guide" *CloudCap Technology*, Hood River, OR, July 2004.

[29] Valenti, M., Schouwenaars, T., Kuwata, Y., Feron, E. and How, J. "Implementation of a manned vehicle - UAV mission system" *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, RI, August 2004 (AIAA 2004-5142).

[30] Watson, N., John, N., and Crowther, W. "Simulation of unmanned air vehicle flocking" *IEEE Theory and Practice of Computer Graphics Conference*, Birmingham, UK, June 2003, 130–137.

[31] Wharington, J. "Autonomous control of soaring aircraft by reinforcement learning" *Royal Melbourne Institute of Technology*, Melbourne, Australia, 1998.